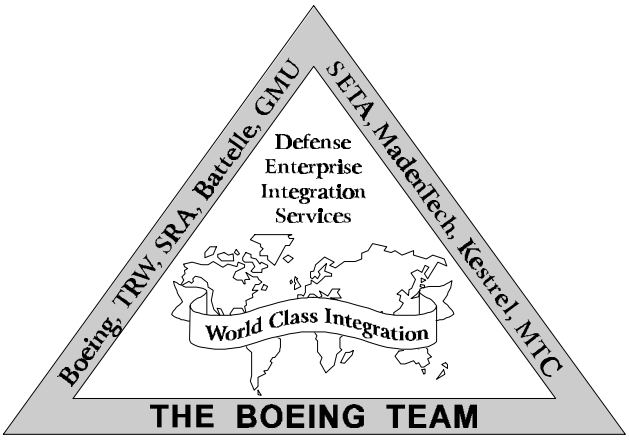
Defense Enterprise Integration Services
Joint Requirement Analysis and Integration Directorate

Contract Number: DCA100-94-D-0016
Delivery Order Number: 0048

GCCS/JOPES Database and Applications Phase IV

Feasibility and Applicability of Web Technologies in the GCCS Environment

048D028



9 July 1996

Prepared by:
Boeing Information Services, Inc.
7990 Boeing Court, CV-47
Vienna, Virginia 22183-7000
(703) 847-1095

Unclassified

TECHNICAL SUPPORT FOR
DEFENSE INFORMATION SYSTEMS AGENCY
ENTERPRISE INTEGRATION DIRECTORATE

DELIVERY ORDER FOR GCCS/JOPES DATABASE AND APPLICATIONS PHASE IV

FEASIBILITY AND APPLICABILITY OF WEB TECHNOLOGIES IN THE GCCS ENVIRONMENT

9 JULY 1996

TABLE OF CONTENTS

TITLE **PAGE**

TABLE OF CONTENTS (CONT'D)

TITLE **PAGE**

TABLE OF CONTENTS (CONT'D)

TITLE **PAGE**

APPENDICES

TITLE **PAGE**

LIST OF FIGURES

LIST OF TABLES

TITLE **PAGE**

SECTION 1 - OVERVIEW

1.1 INTRODUCTION

This technical report supports the objective in Task 11 of the Joint Operations Planning and Execution System (JOPES) Database Migration Technical Proposal, FY95, revised 24 August 1995, under contract number DCA100-94-D-0016, ". . . [to] prototype distributed processing as a means of supporting efficient database access and provide a written report of the results." It proposes transitioning the Global Command and Control System (GCCS) from the closed Worldwide Military Command and Control System (WWMCCS) mainframe to a more open client/server environment. Furthermore, it specifically embraces the Government's desire to seek Commercial Off-the-Shelf (COTS) technology solutions whenever possible. This report evaluates the feasibility of implementing GCCS Database applications in a World Wide Web (WWW) environment using Web technology, or more specifically WWW application development tools.

There are two main sections in this technical report. The first section summarizes the process and technical analysis Systems Research and Applications (SRA) Corporation used to develop a Defense Initiative Infrastructure (DII) prototype using Web application development tools. The second section contains a detailed analysis of this Web technology, including requirements derivation; analyses of the considered products; prototype description and findings; evaluations based on product analyses and prototype findings of an individual product's abilities to satisfy requirements; and overall conclusions and recommendations.

1.2 SUMMARY

The primary goal of this Feasibility and Applicability of Web Technology in the GCCS Environment study is to evaluate the feasibility of implementing GCCS Database applications using Web development tools. The primary and initial goal of the study was to identify and evaluate the Web development tools that could provide access to the GCCS Database. The study also had a secondary objective, to analyze the capabilities of Web Development tools to determine if application development time can be decreased while increasing application functionality.

A major component of this study was the development of a Web-based Scheduling and Movement (S&M) prototype. The main purpose of developing the prototype was to evaluate the development tools and the Web-based client-server database applications generated. Each Web-based development tool was used to construct one or more server applications to access the GCCS/JOPES Database. These applications performed operations common to most database applications: retrieval, update, and rollback transactions. As the study developed, it became clear that HyperText Markup Language (HTML) was incapable of providing the necessary functionality for presenting information at the client's browser. Subsequently, a Graphical User Interface (GUI) was developed using the Java programming language to provide the necessary functionality. This provided additional insights into using the Java programming language for developing Web-based applications.

To determine the feasibility of Web-based GCCS Database applications, three Web-based development tools were selected: Oracle's WebServer (Version 1.0 and 2.0), NeXT's Objects Enterprise, and Netscape's LiveWire. These tools were used to develop Web-based applications that access the GCCS/JOPES Database and interface to the user through a Java applet.

1.2.1 Scope and Methodology

The first step in the analysis was to identify the requirements that a successful Web-based development tool and the subsequent Web-based application must satisfy. The requirements identified were separated into two categories: functional and non functional areas. The functional areas are:

- **Access to GCCS/JOPES Core data located in an Oracle7 database**. Additionally the tool must provide access to data in Sybase and Informix databases.

- **Process distribution**. To provide system design flexibility, applications should be coded in such a manner that separates them from one another and the Database Management System (DBMS) and presentation layers.

- **Product platform independence**. The Web-based development tool and the resulting product must be supported by specific hardware and Operating System (OS) server environments contained in the DII Common Operating Environment (COE).

- **Product interoperability**. The Web-based development tool and the resulting product must work with the chosen DII COE Web browser, Netscape 2.0.

- **Configuration management**. The Web-based development tool must support, in an efficient manner, the configuration management of new versions of the Web-based development tools and the resulting product at both the client and server level.

- **Replicate GCCS Database Application's functionality**. The Web-based development tool should replicate or increase the functionality of S&M's Add Air Carrier thread. The functionality should be provided without the development of additional code or applications.

The non-functional areas are:

- **Security**. The Web-based development tool must provide authentication and authorization when logging into the system.

- **Fault tolerance**. The Web-based development tool should provide the capability to recover from catastrophic failures. In the case of transactional modifications and data retrievals, the system must return to a stable state after the failure.

- **Performance**. The product, produced by the Web-based development tool should perform as well or better than the current implementation.

- **Cost**. The Web-based development tool should produce GCCS applications in a manner that saves development time, development cost and maintenance costs.

The next step was to select candidate Web-based development tools. The Web is an immature and constantly changing environment with new products and releases of current products appearing almost daily. With that in mind, SRA chose not to perform an exhaustive study of Web-based application development tools. Instead, SRA selected leading products that showed promise for satisfying all of the GCCS requirements. The products selected were:

- **Oracle's WebServer** (OWS) includes a HTTP server called the Web Listener, a Common Gateway Interface (CGI) stub called the Oracle Web Agent and the Oracle7 database server. Applications are written in Procedural Language/Structured Query Language (PL/SQL) which are compiled and stored in an Oracle7 database.

- **NeXT's WebObjects Enterprise** includes the WebObjects Framework (WOF) and Enterprise Objects Framework (EOF). The WOF provides an environment to write Web-based applications, while EOF provides a library of predefined classes used by applications to access the database. Applications are developed in WebScript and Objective C.

- **Netscape's LiveWire Beta 2** includes the LiveWire compiler and a server extension. Applications are developed in JavaScript, a scripting language developed by Netscape.

Each of the products selected produce applications that reside on the server. An additional prototype component was developed to serve as the GUI for the Web-based applications. This component was developed in the Java programming language as an applet.

The final analysis step was testing the prototype against the requirements, collecting the results and determining a final recommendation. The final recommendation is contained in this paper.

1.2.2 Analysis

An important aspect of SRA's analysis was the prototype developed using each candidate Web-based development tool. The prototype provided the opportunity to gain hands-on experiences with the different products and form conclusions that would not have been available or observable using only a paper analysis. SRA selected the Add Air Carrier thread of the S&M application. This thread was prototyped because it incorporates a subset of the typical operations performed by GCCS Database applications. The thread consisted of three functions, OPLAN Search, Geoloc Search, and adding an Air Carrier. In addition, a Java applet was developed that functioned as the GUI for the prototype. The client and server machines were connected by means of a Wide Area Network

(WAN), which emulated the Secret Internet Router Network (SIPRNET) connections nearing network speeds.

1.2.2.1 Oracle's WebServer. The OWS portion of the prototype consisted of three PL/SQL programs, the Oracle Web Listener/Web server, and a Java GUI applet. A PL/SQL program was developed which implemented the business rules layer for the Geoloc Search, Oplan Search, and Add Air Carrier functions..

The OWS required a specific Web server, the Oracle Web Listener, and an Oracle database to store the PL/SQL programs. The Oracle Web Listener provided the capability to authenticate the user and this functionality was activated. The OWS provided support for paging of query results. However, the method used to provide the paging involved running the query for each request and only returning a portion of the results, the next page.

The prototype showed several weaknesses and strengths of the OWS product. The major disadvantage to the OWS prototype is its reliance upon a specific Web server, the Oracle Web Listener and requiring applications to be developed using PL/SQL. While PL/SQL provides a stable, platform independent, and easy to learn tool, it must be maintained in an Oracle database The OWS was the most mature of the candidate products and was simple to learn. The OWS prototype required the least amount of development time and exhibited excellent database access capabilities. Performance tests showed the OWS to be the most responsive of the three Web-based development tools when accessing an Oracle database.

1.2.2.2 NeXT Web Objects Enterprise. The NeXT Web Objects Enterprise (WOE) portion of the prototype consisted of two Web-based applications, the Netscape Commerce Server and a Java GUI applet. The Web-based database applications, Geoloc Search and Add Air Carrier, were developed using WOE. Oplan search was not developed. These applications implemented the business rules and data Input/Output (I/O) layers in Web Script and Objective C.

The WOE solution did not require a specific commerce server and the Netscape Commerce Server was used. The Netscape Commerce Server provides secure communication by means of authentication. However, the certificate required to activate authentication was not obtained. WOE was unique among the Web-base application tools because it provided support, out of the box, to keep state information for the user within the server. This capability was used to implement Geoloc Search query paging in the optimizing, CPU and network loads. The query was executed once, but only returned a page of data when it was requested.

The prototype development showed several strengths and weaknesses with the WOE product. WebScript was best suited for the portion of the application that interfaces to the GUI. Those portions of the application which involved more complex processing required development in Objective C. Using Objective C to implement the business rules also revealed some problems. The Objective C language, an object-oriented extension to the C language, was easy to learn, but the development environment and the process to incorporate it into the WOE application was poorly documented. WOE interacts with the Oracle database by dynamically generating SQL queries. This

generated slower response times than the compiled SQL statements used by OWS. The WOE however does have the capability to maintain state information.

1.2.2.3 Netscape's LiveWire. The Netscape LiveWire portion of the prototype consisted of three Web-based applications, the Netscape commerce Server, LiveWire compiler, LiveWire Server extension, and a Java GUI applet. A LiveWire application, written in JavaScript was developed for Geoloc Search, Oplan Search, and Add Air Carrier which implemented the business rules layer.

LiveWire required Netscape's Commerce Server, currently Version 1.2. During the prototype development, Netscape informed SRA that LiveWire would not be supported for Netscape's Commerce Server, but would require Netscape's Enterprise Server which would provide additional functionality. The LiveWire development environment did not provide a means to maintain state information. Therefore, large query retrievals, such as Geoloc Search, were implemented to perform the query once and return all data retrieved.

The prototype showed several strengths and weaknesses with the LiveWire product. JavaScript is a dynamically binding language that checks object references at runtime. Simple errors such as misspellings would not be caught by the compiler and thus cause additional iterations in the code-compile-test-debug cycle. The LiveWire development environment was simple to use and productive but lacked sufficient documentation. The LiveWire prototype was developed rapidly requiring slightly more development time than the OWS prototype. A major problem was encountered using the LiveWire product. When too many (i.e., more than 12) LiveWire applications were registered, the servers capability to execute any CGI program was stopped. Additionally, a problem was encountered with maintaining registered applications. When an application required updating, the server had to be restarted before the application would run again.

1.2.2.4 Java. The primary focus of this analysis was to analyze the ability of Web-based development tools to provide access to Oracle databases. As the analysis progressed it became apparent that applications with this type of functionality required a more sophisticated interface than could be provided by HTML pages either statically or dynamically generated. Java, Sun's new platform independent programming language, was selected to develop the GUI for the Web-based applications. In this paradigm, a program called an applet is downloaded from the server into the client's browser when the client selects an Uniform Resource Locator (URL). The applet byte code is then interpreted and controls the interactions with the user.

The prototype showed several weaknesses and strengths with the Java programming language. It's primary weakness is its immaturity which can be found in the level of available documentation, and non-availability of third party products for GUI design, development libraries, and environments. There were also several bugs in the implementation. The Java development environment was simple to learn for a non-novice programmer and incorporated many additional features into the language which are only provided as third party products in the traditional development environments. Additionally, Java did develop platform independent code. The GUI developed was downloaded into multiple platforms and executed successfully with no modification.

1.2.3 Product Comparisons and Recommendations

The next step in analyzing the solutions is comparing each solutions's ability to satisfy each requirement. This comparison is summarized in Table 1-1 Web-Based Development Tools Product Comparisons. Each solution is give a rating based on it's ability to satisfy a specific requirement. A € - indicates that the solution does not fully satisfy the requirement or that significant development is required to satisfy the requirement. A € indicates that the solution satisfies the requirement. A € + indicates that the product more than satisfies the requirement. **N/A** indicates that the requirement was not applicable to the product.

Table 1-1: Web-Based Development Tools Product Comparisons.

Requirement Area	Oracle WebServer	NeXT WebObjects	Netscape LiveWire	Java
Access to GCCS and JOPES Data	€ -	€	€	**N/A**
Process Distribution	€ -	€	€	€
Product Platform Independence	€	€	€	**N/A**
Product Interoperability	€	€ +	€	€
Configuration Management	€	€	€ +	€ +
Duplication of GCCS Database Application's Functionality	€	€	€	**N/A**
Security	€	€	€	**N/A**
Fault Tolerance	€	€	€	**N/A**
Performance	€ +	€	€	€ -
Cost	€	€ -	€	**N/A**
Overall Rating	€	€	€	€

The WWW technology landscape is rapidly changing, and new products are continually emerging. Additionally, Web technology as a whole is immature. SRA feels that the rapidly changing Web landscape and the immaturity of many Web products represent risk.

Deploying applications on the WWW is very much a network-centric solution, and as such, network response time becomes critical. In SRA's prototype, the amount of available bandwidth did not greatly affect performance. In particular, the amount of time it took to download the Add Air Carrier thread did not increase significantly at lower bandwidths. However, as Java applets become more complex and encompass more functionality their size will also increase, making available bandwidth a more crucial parameter.

Another potential problem area when deploying applications on the WWW is security even if the network on which the application is deployed is an Intranet. There is a long list of security mechanisms available to an administrator such as encryption at the hardware and software levels and client authentication.

There are many benefits with deploying applications on the WWW as well. The Web has the potential to bring platform independent applications to the desktop. This potential is already largely filled in terms of clients. As platform independent languages like Java or JavaScript become more widely used and supported by HTTP servers', this potential will be available at the server level. Not only does this platform independence facilitate the idea of distributing processing to where it makes the most sense, it also tremendously eases the configuration management workload.

As a result of performing this evaluation, SRA has formulated several recommendations. First, SRA does not believe that the state of Web technology is ready to field mission critical applications for use by power users. Instead, SRA feels that the current state of Web technology lends itself to providing application functionality, especially database browsing, for the casual user. For heavy users of GCCS, SRA feels the performance that can be provided by current Web back end database solutions is insufficient and the functionality that can be provided by a GUI developed in Java and HTML is insufficient. This does not mean that it is not a good idea to begin the migration of GCCS Database application threads. On the contrary, now is a perfect time to become more knowledgeable about the details of deploying applications on the WWW.

Programming expertise is not SRA's primary concern with Java. Instead, SRA is most concerned about Java's performance and security. Sun claims that for most applications, Java provides more than adequate processing and performs only slightly worse than machine specific code. SRA feels that these two areas must be investigated more thoroughly in order to substantiate this claim. SRA's performance testing, although by no means a complete analysis of Java's performance, indicate that Java may have some serious performance problems when processing large amounts of data. Java's security needs to be investigated further as well. In the short time it has been out, security holes have already been discovered.

Other areas that SRA feels need to be evaluated more in-depth include HTTP servers and Web browsers. HTTP servers play a significant role in terms of performance and security for the overall system. The current widely used Application Programming Interface (API) is the CGI, but this interface lacks the performance to support complex applications and high transaction rates. HTTP vendors are developing their own proprietary more performant APIs such as Netscape's NSAPI and Oracle's Web Request Broker (WRB) API. It is important to study this emerging trend from both a technical and a business standpoint in order to choose the most appropriate API in which to invest. HTTP security mechanisms must be studied and tested to determine which ones will provide the best level of security. In the case of Web browsers, SRA recommends that the Government study the developing market of plug-ins and helper applications that third party vendors are developing for Web browsers. In this way, the idea of using COTS products may be advanced. Additionally, if more advanced security features are desired, it must be ensured that the Web browser can support them. Another point that relates to Java but pertains to both HTTP servers and Web browsers is the ability to cache Java applets to improve performance. This concept should be investigated and tested.

SRA wants to stress that new web application development tools have been developed during SRA's study and more will be developed in the future. With this said, however, SRA feels that the direction the government should take is to monitor the progress of and evaluate these emerging tools. One specific tool to monitor are Fourth Generation Language (4GLs) that provide the ability to produce code using C, C++ and other hardware and OS specific programming languages for better performance, interface quality, and the ability to produce code using platform independent languages such as Java. The latter applications may be less performant than the former, but they will facilitate process distribution and configuration management. At a minimum, if the government is to proceed with Java development, it must study Java development tools. As an accompanying thought, SRA would also like to point out the emergence of middleware products that can convert hardware and OS specific code such as Ada 95 into platform independent bytecode.

SECTION 2 - ANALYSIS

2.1 INTRODUCTION

This section follows SRA's repeatable process for evaluating technology. First, important goals and assumptions that shaped the evaluation are presented. Then, SRA discusses the functional and non-functional requirements, followed by a detailed analysis of the Web application development tools studied in this evaluation: Oracle's OWS 1.0 and 2.0, NeXT's WOE and Netscape's LiveWire Pro. Additionally, the analysis addresses the Java programming language's ability to serve as the development language for the GUI. This analysis also encompasses a prototype that uses each of the products to implement a portion of GCCS functionality. Next, SRA rates the products' abilities to satisfy the different requirement areas before concluding the evaluation with observations and recommendations.

2.1.1 Evaluation Goals and Assumptions

The primary goal of this study is to evaluate the feasibility of implementing GCCS Database applications using Web development tools. A secondary objective is to analyze the capabilities of Web application development tools to see if they can decrease application development time while increasing application functionality by providing out of the box capabilities. In keeping with this goal, SRA implemented the individual prototypes in a way that made use of the capabilities to which each product best lent itself. Some important assumptions that were made include:

- This paper is not intended to be an evaluation of HyperText Transfer Protocol (HTTP) servers, but some discussion about them is included because they affect the vital areas of performance and security.

- This paper is not intended to address the client in the sense that Web browsers are not evaluated. The Netscape Navigator Web browser is used in the prototype.

- This paper is not intended to be an exhaustive study of all Web application development tools. Instead, SRA chose three leading products that showed promise for satisfying all the requirements for GCCS.

- The focus of this evaluation is on Web application development tools that enable the development of interactive, Web-based database applications. In particular, the ability to access databases in a heterogeneous environment is important. An inseparable part of developing an interactive application, however, is the development of a GUI. As a result, this evaluation also addresses this aspect, although to a lesser degree than the database access portion.

2.2 REQUIREMENTS

The first step in the evaluation of Web application development tools is the derivation of requirements. A solution using Web technology must satisfy several requirements in order to help accomplish the primary goal of GCCS: the support of the global planning and operations of the United States' Armed Forces. These requirements fall into several functional and non-functional requirement areas. Functional requirement areas include access to GCCS and JOPES data, process distribution, product platform independence, product interoperability, configuration management, and replication of existing GCCS Database applications' functionality. Non-functional areas include security, fault tolerance, performance, and cost. The following sections discuss each of these functional and non-functional areas.

2.2.1 Functional Requirements

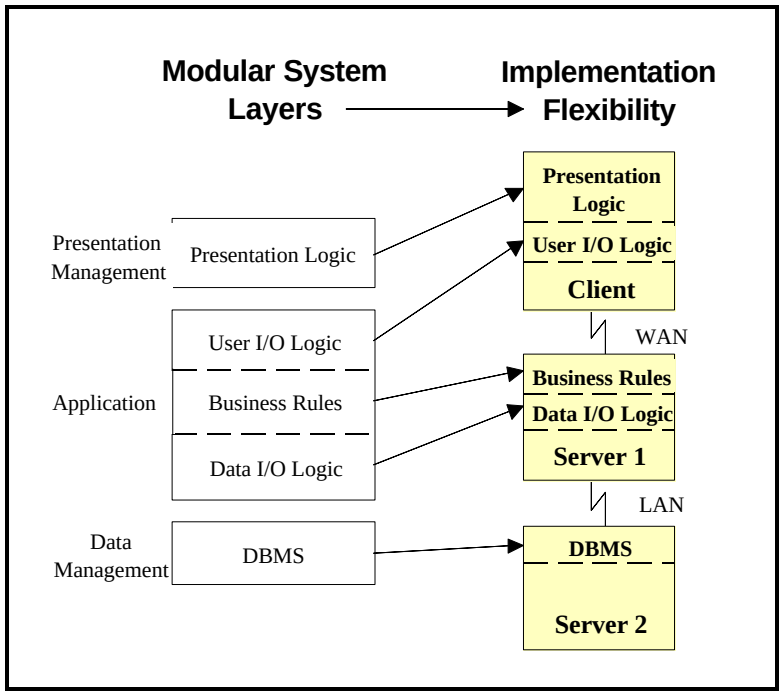
Functional requirements define functionality that is necessary to perform the daily operations that a system is supposed to support. Any implementation of a GCCS distributed database application in the DII environment using Web technology must satisfy several functional requirements to provide the basic functionality that is a part of GCCS applications.

2.2.1.1 Access to GCCS and JOPES Data. The principal requirement that must be satisfied by the Web technology is the ability to access GCCS and JOPES data residing in the GCCS/JOPES Core Database, an Oracle7 database. In this instance, access involves two types of operations: modifying and retrieving data. In terms of modifying data, any Web-based database application must support the ability to perform transactional modifications which may consist of multiple database inserts, updates and deletes specified by a user. In the event that any one of these inserts, updates or deletes fails, the Web-based application must be capable of nullifying the entire transaction to ensure that no portions of the transaction are applied to the database. Retrieving data is the second aspect of access to GCCS and JOPES data. For this requirement, the Web-based application must support the retrieval of data from the GCCS/JOPES Core Database as specified by users. For both transactional modifications and retrievals, the Web-based application must allow multiple users to access the same data simultaneously.

In addition to the primary requirement of accessing Oracle7 data, a secondary requirement is that the Web-based application must be able to access data residing in Sybase and Informix databases. Ideally, access to any of the databases, including Oracle, should be accomplished using standard interfaces such as the Open Database Connection (ODBC) standard or X/Open's XA standard. Minimally, the Web-based application will access these databases using the databases' well documented and published proprietary Call Level Interfaces (CLIs): Oracle Connection Interface (OCI), DB-Lib/CT-Lib and ESQL (Embedded Structured Query Language)/C for Oracle, Sybase, and Informix respectively.

Using a combination of Oracle7 environment variables and the SQL*Net transportation protocol, current GCCS Database applications allow an administrator, at installation time, to select a target database on which operations should be performed. In this instance, the administrator can only change databases by modifying start-up scripts or shell environment variables. These methods are outside the scope of the current applications and can only be accomplished through a non-

programmatic, manual process.
Consequently, a requirement is
to allow users, if they so desire,
to specify programmatically which
ly a database on should be
operations the user,
performed. If not wish to
however, does database, then
specify a must use a
the application default database.



2.2.1.2 Process Distribution.

An important attribute that the
chosen Web application
development products must
support is the ability to
distribute processing in a
network-wide manner.
Inherent to this requirement is
the need to be able to code application layers in a manner that separates them from one another and from the DBMS and presentation management layers. Applications can generally be thought of as consisting of three layers: user I/O logic, application logic and data I/O layers. The user I/O logic layer interfaces with the presentation management layer to provide users with a GUI that allows them to input and display data. The application logic layer consists of programming logic and business rules, while the data I/O layer interacts with the DBMS to perform database queries and updates on behalf of the user. In short, the Web technology must support the ability to distribute system layers to any computer resource in GCCS as necessary. It is important for the Web technology to support this modular approach because it enables the implementation of applications that are not hampered by physical boundaries. Instead, applications will be able to make better use of a network's available resources. Figure 2-1 illustrates this idea.

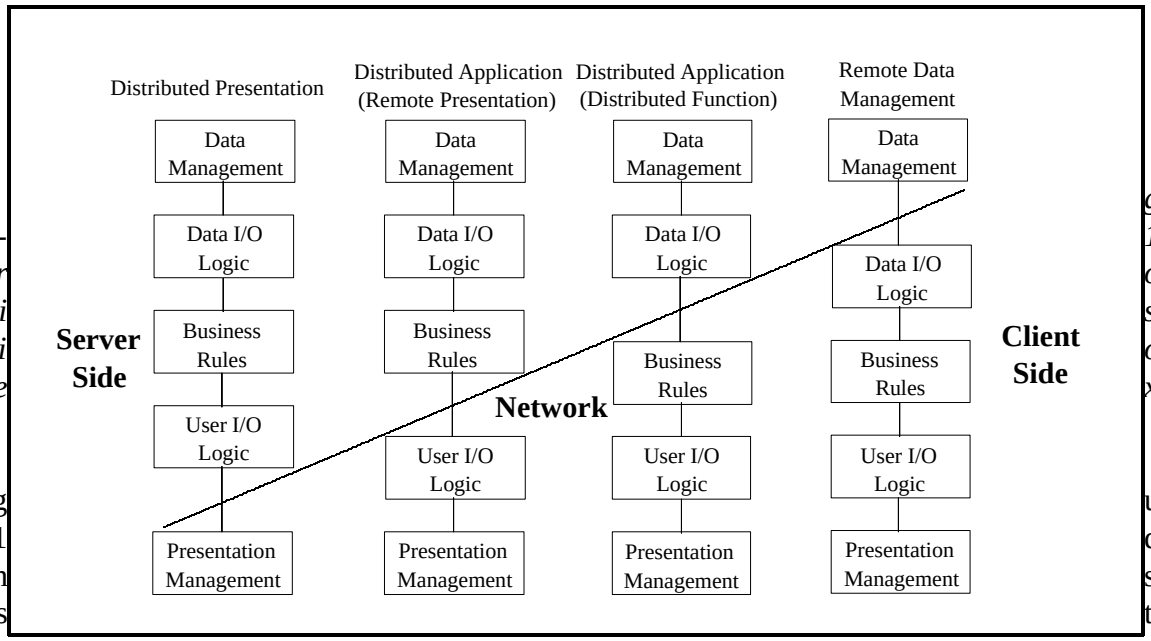


Figure 2-1: Process Distribution Flexibility.

Figure 2-1 demonstrates that modular components provide system design flexibility. Notice that logical layers can be distributed to multiple servers, thus increasing the computing power available to the system. Additionally, it is possible to choose the Local Area Network (LAN)/WAN configuration that makes the most sense. The distribution of processing shown in Figure 2-1 shows the system layers distributed in a specific configuration. Figure 2-2 shows a more generic view of how processing can be distributed.

Figure 2-2: Process Distribution.

Figure 2-2 illustrates process distribution from a server side/client side viewpoint. It is important to understand that the layers on the server side could be distributed to different servers

connected via WANs and LANs. Furthermore, another implementation option is to place all the processing on the same machine.

In order to achieve the previous requirement of a modular architecture, any developed code must be platform independent. Although it would be ideal for the code to be completely platform independent, this goal is not realistic. It is currently only necessary for developed code to be operable in Solaris 2.x, HP-UX 9.x, and Windows NT OS environments running on SPARC, HP-PA RISC, and Intel hardware platforms respectively. These OS and hardware platforms must be considered because they are a part of the Version 3.0 of the DII COE. Another accompanying requirement is that the data I/O code must be able to access a database that is located either locally to where the code is being executed or remotely on another server. This access should be transparent to the user.

2.2.1.3 Product Platform Independence. It was just discussed that any developed code must be supported by specific hardware and OS server environments. This requirement also holds true for any product that are used in the creation of database access solutions. More specifically, these products must be able to operate within the Solaris 2.x, HP-UX 9.x, and Windows NT OS environments running on SPARC, HP-PA RISC, and Intel hardware platforms respectively.

2.2.1.4 Product Interoperability. At a minimum, any chosen Web application development tools must be able to interoperate with the Netscape 2.0 Web browser because it has been chosen as the DII COE Web browser. Additionally, the products' support of open standards as discussed in Paragraph 2.2.1.1 is integral to achieving interoperability.

2.2.1.5 Configuration Management. An important attribute of all systems that are fielded in the GCCS environment is configuration management. A Web-based solution is no exception to this fact and must support ways of performing configuration management in an efficient manner. When evaluating configuration management in a distributed, multi-user environment, it is important to consider configuration management at the client level and at the server level. Furthermore, there are several aspects that comprise configuration management. First, there is the configuration management, or installation and upgrade, of the web application development tools themselves. When new versions of these products are available, they need to be installed and configured. Second, is the configuration management of the applications themselves. This area involves updating applications with code corrections or entirely new baselines. The third type is the administration of infrastructure products such as the Web browser. In this paper, SRA addresses the first two aspects, but the third one is not evaluated as these infrastructure products fall out of the realm of this study. Configuration management requirements should be addressed by the modular architecture of the system. An additional requirement is to allow a user, who is initially logging into the system, to choose the baseline with which to work. If the user does not wish to specify a baseline, then the system should default to the most recent baseline.

2.2.1.6 Replicate GCCS Database Applications' Functionality. A primary goal of this evaluation is to determine the effectiveness with which Web technology can implement existing GCCS software functionality. For this evaluation, the Web technology must provide a solution which replicates or increases the functionality offered by S&M's Add Air Carrier thread's functionality. Although the

JOPES S&M Software Requirements Specification (SRS) delineates these requirements in detail, the Web technology evaluation emphasizes the fulfillment of the following requirements:

- The ability to add, based on user selection and inputs, air cargo/passenger carriers to the GCCS/JOPES Core Database

- The implementation of a minimal set of validation rules for the GUI to include validation that field data entries match field data types and verification that entered dates are valid, not-null fields have valid entries and itineraries are sequential

- The execution of the Add Air Carrier business logic and the insertion of records into the following GCCS/JOPES Core Database tables:

  - OPLAN_Carrier which contains the OPLAN/carrier pair
  - Carrier which contains carrier information
  - Carrier_Itinerary which contains itinerary records based on user input
  - Send_Queue which contains SCHDET and SCHPET transactions

- The ability to update local and remote databases in a way that is transparent to the user.

In addition to this functionality, the Web-based database application must also provide the Geolocation Search functionality, a subportion of the Add Air Carrier thread. In particular, the user must have the ability to submit a query for geolocation codes in accordance with the JOPES S&M SRS.

Finally, the Web-based database application must support the current and future GCCS permissions schemata. The current schema for GCCS 2.x consists of the following types of permissions:

- OS level permissions which consist of read, write and execute privileges on system files. These permissions are associated with UNIX User IDs and control access to various GCCS executables.

- Oracle7 table permissions based on the Oracle7 User ID. These permissions control access to GCCS/JOPES Core Database tables based on the user's role (i.e., users can access all, none or a portion of the GCCS/JOPES Core Database tables based on their assigned roles).

- User Operational Plan (OPLAN) permissions based on OPLAN series permissions and OPLAN specific permissions. Working in conjunction, these two permissions determine whether a user has access to an OPLAN based on the GCCS User ID and on the particular OPLAN that is being accessed. They are implemented via the application accessing Oracle7 tables to determine if the user has access to the OPLAN based on GCCS User ID and on OPLAN availability.

- Functional permissions that define the functions a user may perform on a particular OPLAN. Much like user OPLAN permissions, they are implemented via the application accessing an Oracle7 table to determine which functional permissions are associated with the user.

In future releases of GCCS, the permissions scheme for accessing information from the GCCS Core Database will consist of the following types of permissions:

- OS level permissions and Oracle table permissions as previously described.

- User OPLAN permissions and functional permissions enforced on all data accesses of any GCCS/JOPES Core Database table. Although these are the same permissions used in the current scheme, they will be implemented via Oracle views which use the user's Oracle ID as well as access control information from other Oracle tables to verify data access privileges.

Any database application in the DII environment must be able to enforce these permissions in the future.

2.2.2 Non-Functional Requirements

Non-functional requirements work in conjunction with functional requirements to define completely the guidelines for developing a Web-based database application for the DII environment. Non-functional requirements refer to system attributes that enhance the functionality of the system. It must be understood that in no way are non-functional requirements necessarily less important than functional requirements, and in many cases, the non-functional areas may turn out to be the discerning factors between different vendors' products.

2.2.2.1 Security. Security requirements include authentication and authorization when logging into the system. The system must authenticate the user and verify the user's authorization level prior to allowing the user to perform operations on the system. The system should require the user to authenticate only once to the system. Access is then granted to the user based on groups or roles determined by the user's identity. Unauthorized access during login should result in an appropriate error message as well as notification of the attempted access to the system. If unauthorized access occurs during an attempted database operation, the system should notify the user with an appropriate error message and immediately discontinue the operation. If a user is properly authenticated, the system should allow the user to perform a finite set of business operations on the database using any DII Web-based database application.

2.2.2.2 Fault Tolerance. Any database access solution implemented in the DII environment must have the ability to recover from catastrophic failures that occur in the data management, data I/O and business rules layers. More specifically, in the case of transactional modifications and data retrievals, the system must return to a stable state after the failure so that the modification or query may be retried. In the event of failures in the presentation and user I/O logic layers, the Web-based application must report an appropriate error message to the user and return control gracefully to the

calling process. Additionally, in the event of a database server failure, the system must be capable of rerouting the user to an available database that contains the OPLAN on which the user was working at the time of the failure.

2.2.2.3 Performance. The primary performance requirement for the development of a DII distributed database application is to provide improved or at least comparable performance to the performance of the existing implementation of the Add Air Carrier thread. A major focus of this requirement is the performance of the application over lower speed links such as 64 kilobits per second (kbps). Consequently, the application must minimize the amount of network traffic associated with any Web-based solution. This requirement is particularly crucial in a WAN environment such as GCCS where bandwidth is at a premium.

2.2.2.4 Cost. One of the primary goals of this evaluation is to produce a Web-based database application in a manner that saves development time and cost and maintenance cost. The evaluation of any Web application development tool's ability to fulfill these requirements is obviously subjective and difficult to quantify. With this fact in mind, SRA's engineers evaluate these requirements based on experience gained during the prototype effort. Another important aspect is the cost of actually purchasing the Web products. Although the government has not provided SRA with any specific guidelines concerning the cost of the products, SRA includes information on this area in this paper.

2.3 PRODUCT ANALYSES

After the development of functional and non-functional requirements, the next step in the evaluation of the Web application development products is analyzing each candidate's capability to fulfill these requirements. This section presents an analysis of Oracle's OWS 1.0 and 2.0, NeXT's WOE 1.0, and Netscape's LiveWire Beta 2. These product analyses coupled with the prototype findings detailed in Paragraph 2.4 constitute the reasoning behind the evaluations and ratings presented in Paragraph 2.5 and ultimately lead to the recommendations and conclusions contained in Paragraph 2.6.

2.3.1 Background WWW Technology and Concepts

Before proceeding to the analysis of the individual products, it is necessary to provide some background on WWW technologies. This background information provides a foundation for the more detailed discussions on each product in the upcoming sections.

2.3.1.1 Uniform Resource Locator. URLs provide a standard way of representing hyper media links and links to network services. In the WWW, URLs can reference files and other resources available over several different protocols such as HTTP, File Transfer Protocol (FTP) and Wide Area Information Search (WAIS). The general format of a URL is as follows:

```
scheme:scheme_dependent_information
```

where `scheme` is HTTP, FTP, WAIS, etc. The `scheme_dependent_information` is dependent upon the particular scheme being used, but in most cases, it consists of the name of the Web server and the full path to the requested file. The following is an example of an HTTP URL:

```
http://www.sra.com:80/srahp/dqe.html
```

where `http` is the scheme and the rest of the URL denotes the scheme_dependent_information. `www.sra.com` is the hostname or Internet address of the Web server that is being accessed and `80` is the transport protocol port (TCP) on which the HTTP server is running. `srahp/dqe.html` represents the path to the requested file, `dqe.html`. In the case of HTTP URLs, the path may be absolute or relative depending on how the HTTP server is configured.

2.3.1.2 HTTP. HTTP is the application-level protocol which Web clients and servers use to communicate with one another. HTTP follows a request/response model. In this environment, clients send HTTP requests to an HTTP server. The server interprets the request by reading the URL associated with the request. After interpreting the URL, the HTTP server responds by returning the text and any other media referenced by the link in the URL.

In the request/response model, the client is responsible for establishing the connection to use for transmitting the request, and the server is responsible for closing the connection after it has returned a response to the client. HTTP is stateless which means there is no continuity from transaction to transaction. Once a transaction is completed there is no recollection of it occurring. HTTP uses TCP/Internet Protocol (IP) connections for its transport and routing mechanisms. This

fact, however, does not imply that HTTP cannot use other types of connections. In fact, the HTTP request and response structures can be mapped to any protocol that guarantees a reliable transport mechanism.

An important occurrence during the previously described connection is protocol negotiation. During protocol negotiation, HTTP clients and servers exchange information about the types of data they can transfer and interpret. Furthermore, the client may express preferences for certain types of data. If the HTTP server has data stored in more than one format, it can choose a format the client supports or prefers. For example, a document's in-line image may be stored in either the Graphical Interchange Format (GIF) or the Joint Photographic Experts Group (JPEG) format, and based on the formats supported and preferred by the client, the listener will choose a format. This same idea can be applied to the national language of a document. The same document may be displayed as one of many national languages based on the preference expressed by the client.

In short, HTTP serves as a communications link between user agents such as Web browsers and other Internet protocols such as FTP. HTTP also provides the Web with its multimedia capabilities because it supports the retrieval and display of text, graphics animation and the incorporation of sound.

2.3.1.3 HyperText Markup Language. HTML is the standard language for creating hypermedia documents on the Web. HTML documents are simply American Standard Code for Information Interchange (ASCII) files with formatting codes, or tags, that contain information about document layout such as image placement and text styles. It is the browser's responsibility to interpret the formatting tags. Pages may look slightly different from browser to browser because each browser may interpret a particular code in a slightly different way. The overall look and feel of the document, however, is consistent from browser to browser. The fact that the HTML tags are interpreted by the browser allows HTML documents to be served and viewed in a multi-platform environment. The most currently approved version of HTML is Version 2.0, but many browser manufacturers such as Netscape are adding HTML extensions to their browsers that enhance the functionality of HTML.

2.3.1.4 Secure Sockets Layer. The Secure Sockets Layer (SSL) is a protocol designed to provide secure communications between a client and a server using TCP/IP or some other reliable transport protocol. The SSL positions itself between the transport layer and the application protocol, allowing diverse protocols such as HTTP and FTP to make use of SSL. There are three basic principles at the foundation of the SSL:

- All messages are encrypted
- The server endpoint of the connection is always authenticated, and the client endpoint is optionally authenticated
- The integrity of data is ensured.

SSL is a non-proprietary protocol which provides data encryption, server authentication, message integrity and client authentication for TCP/IP connections. SSL gives the server and client

the ability to authenticate each other. An encryption algorithm and cryptographic keys are negotiated before any data is transmitted.

SSL supports secret/private-key and public-key encryption. Both sender and receiver have two related complementary keys, a publicly revealed key and a secret key. When the sender wants to transmit data, the receiver's public key must be used to encrypt the data. Once the data is received, the receiver can decrypt the data with it's own private key. Knowing the public key does not help to deduce the corresponding secret key. The public key can be published and widely disseminated across a communications network. Anyone can use a recipient's public key to encrypt a message, and that recipient uses their own corresponding secret key to decrypt the message. No one but the recipient can decrypt it, because no one else has access to that secret key. Not even the person who encrypted the message can decrypt it.

Message authentication is also provided. The sender's own secret key can be used to encrypt a message. This creates a digital signature of a message, which the recipient can check by using the sender's public key to decrypt it. This proves that the sender was the true originator of the message, and that the message has not been subsequently altered. The sender alone possesses the secret key that created the signature. Forgery of a signed message is not feasible, and the sender cannot later disavow his signature.

2.3.1.5 Common Gateway Interface. The CGI is a specification designed to allow HTTP servers to make use of external resources such as other server programs. As such, it is commonly supported by HTTP servers. By executing CGI compliant programs, HTTP servers are not limited to serving static documents. Instead, HTTP servers are able to provide Web users with additional functionality such as processing data, calculating information and performing data searches.

When a browser issues a request, in the form of a URL, to the HTTP server to execute a CGI program, the HTTP server creates a separate process for this purpose. The HTTP server then maintains interprocess communications with the process for two reasons. The first reason is so that the HTTP server can transmit input from the client to the CGI program. The second reason is so that the HTTP server can receive the CGI program's HTML output from standard output, the place where the HTTP server expects CGI programs to send their output. There are generally two ways that the submitted URL can inform the HTTP server to execute a CGI program. The first way is to configure the HTTP server to recognize files with .cgi extensions as CGI-compliant programs. Another way is to configure the HTTP server to recognize a specific directory as containing only CGI-compliant programs. When a URL is submitted that points to a file in this directory, the HTTP server treats it as a CGI program.

There are two interfaces involved in transferring data between the client's browser and the CGI program. The first interface handles data passed between the client's browser and the HTTP server. The second interface handles data passed between the HTTP server and the CGI program. Data may be passed between the client's browser and the HTTP server using one of three methods: the GET method, POST method or ISINDEX method. The transfer method is specified in the HTML page that is transmitting the data to the CGI program. The chosen transfer method affects the format of the submitted URL. Using the GET method, the user's data is concatenated to the

URL that points to the CGI program. As an example, consider a search program that must search for the term computer. The submitted URL would look like the following:

```
http://www.company.com/cgi-bin/search?term=computer
```

where `term` is a predefined variable from the HTML page. Additional variables, or search parameters, may be concatenated to the URL and separated by an ampersand. Using the POST method, the browser does not modify the URL. Instead, it writes the data to the HTTP server directly. The ISINDEX method is similar to the GET method in that the URL is used to pass the data. For the ISINDEX method, the sample GET method URL would look like the following:

```
http://www.company.com/cgi-bin/search?computer
```

Additional data is concatenated to this URL and separated by a plus sign.

The method selected for passing data between the client's browser and the HTTP server dictates the method used to pass data between the HTTP server and the CGI program. When using the GET method, the data is passed to the CGI program using CGI environmental variables. Table 2-1 contains some of the more important CGI environment variables.

Table 2-1: CGI Environment Variables.

Variable	Contains
REQUEST_METHOD	Method used when accessing a URL on the server
PATH_INFO	Additional path information that the server derives from the URL used to access the CGI program
SCRIPT_NAME	Allows the CGI program to refer the client back to itself
QUERY_STRING	Information to be passed from the client to the CGI program
HTTP_USER_AGENT	The browser the client is using to send the request
PATH_TRANSLATED	Translated version of PATH_INFO provided by server for mapping. This consists of the document root concatenated to the PATH_INFO
REMOTE_HOST	Hostname making the request if it can be determined
REMOTE_ADDR	IP address of the remote host making the request
REMOTE_USER	Used to authenticate user
REMOTE_IDENT	Set to the remote username retrieved from the server
SERVER_PROTOCOL	Name and revision of the information protocol used in the request

Variable	Contains
SERVER_SOFTWARE	Name and version of information server software answering the request
SERVER_NAME	The server's hostname or IP address
SERVER_PORT	Port number on which the server is running

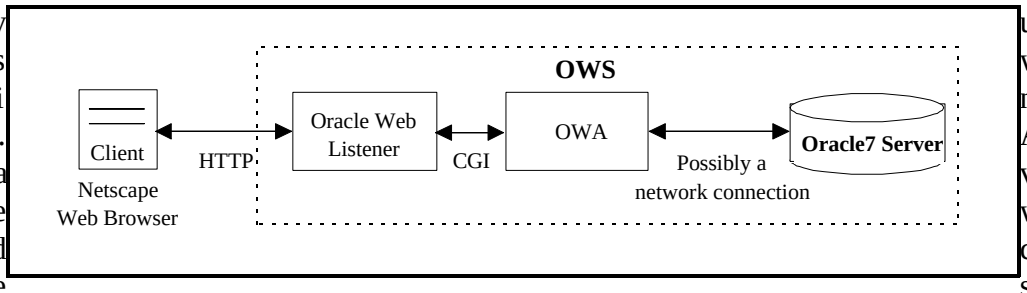
Notice from Table 2-1 that the user's data is contained in the environment variable QUERY_STRING. The remaining environment variables are provided by the HTTP server to pass information about the client's request to the CGI program. One potentially serious drawback to using the GET method is its use of environment variables to pass information because there are limits to their lengths. Therefore, for example, if it was necessary to pass many parameters to the invoked CGI program, it is very possible that the QUERY_STRING variable will not be able to handle all of them. The POST method, although more complex to code, does not suffer from this limitation.

Originally, CGI was designed to handle low-volume requests rather than high-volume server requests and transaction processing loads. In many cases, the current use of CGI is outstripping it original design. As a result, many HTTP server vendors are developing their own API which can interact directly with server functions and bypass the CGI. The Netscape Server API (NSAPI) and Oracle's WRB are examples of these APIs. The main reason to use an API such as the NSAPI or WRB is improved performance over the CGI.

2.3.1.6 State. The concept of state was introduced in the description of HTTP when it was mentioned that HTTP is an inherently stateless protocol. With HTTP, each request from the client arrives at the server with no relation to previous requests. If, for example, the HTTP server executes a CGI-compliant program that returns output to the client, the CGI program then terminates, and there is no memory of the interaction. For many applications, however, it is desirable to maintain a state, or memory, of previous work between user sessions because maintaining state leads to more efficient implementations. State information may be maintained at the client or at the server, but in general, it is preferable to maintain state information at the server rather than the client for several reasons. First, in order for server applications to store and to make use of state information at the client, this information must be transmitted to the client, using valuable network resources in the process. Second, the developers must write additional code to handle the passing back and forth of the state information between the client and the server. Finally, some state information such as a database login cannot be easily stored on the client.

In the case of client side or server side state, if a Web application development tool does not provide out of the box functionality to perform state, developers may find themselves having to write significant amounts of custom code to maintain state. The tracking of state information may or may not be important depending on the functionality that a particular application must provide. In some cases, this functionality will dictate the need for state information, but in other cases, state information will not be needed at all. For this evaluation, a specific use of state management is discussed in relation to the implementation of the Geoloc Search functionality. In particular, state management plays a role in performing the geolocation code query in an efficient manner. Database applications

typically use cursors when performing queries. A cursor is like a virtual table whose rows and columns are specified by the query, and results from the query are stored in the cursor. When implementing the Geoloc Search functionality, it is desirable to be able to associate a particular cursor with a particular user across multiple HTTP sessions. This idea is discussed in more depth in the prototype sections.



2.3.2 OWS

When SRA began its evaluation of Web application development tools Oracle offered the OWS 1.0 as a means for developing Web-based applications. During the course of the evaluation, however, Oracle released its OWS 2.0. This paper discusses each product beginning with OWS 1.0. Furthermore, SRA was able to prototype both products, and this effort is discussed in Paragraph 2.4.1. Many of the points about OWS 1.0 hold true for OWS 2.0, but there are some notable differences that will be made clear.

2.3.2.1 OWS 1.0. The OWS consists of several components: an HTTP server called the Web Listener, a CGI stub called the Oracle Web Agent (OWA), and the Oracle7 database server. Figure 2-3 shows these components and their relationship to one another.

Figure 2-3: OWS Components.

The OWS also offers a developer's toolkit which contains a set of procedures that facilitate the development of applications. The OWS supports the Solaris 2.x, HP-UX 9.x, and Windows NT OSs running on SPARC, HP-PA RISC, and Intel platforms respectively. Additionally, the OWA requires the Web Listener because it is unable to interoperate with other HTTP servers. Each of the WebServer's components will now be discussed in more detail.

2.3.2.2 Oracle Web Listener. It is important to note that HTTP servers are outside the scope of this evaluation. At the same time, however, they must be considered in the evaluation of security and performance as they play an important role in these two non-functional requirement areas. The discussion on Netscape's Commerce Server will follow the same format. The Oracle Web Listener

is a commercial quality HTTP server. As is the case with HTTP, the Web Listener's operation is based on the request/response model. Clients send HTTP requests to the Web Listener, and the Web Listener interprets these requests by reading the URL associated with the request. After interpreting the URL, the Web Listener returns the requested information to the client. The communications between the Web Listener and its clients is accomplished using HTTP 1.0.

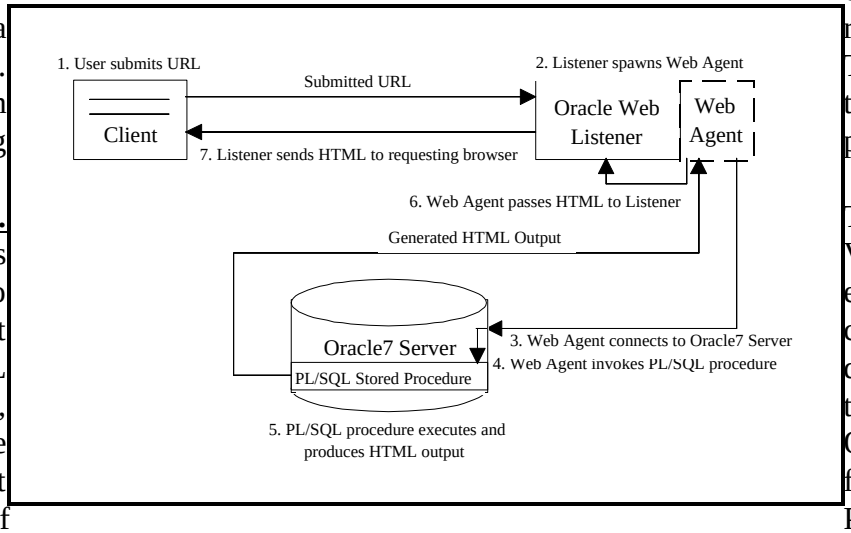
2.3.2.2.1 Performance and Security Features. The Web Listener runs as an asynchronous engine using a single process with a single thread. Whenever a connection is made, the Web Listener uses the already running process and thread rather than starting a new process or thread each time a connection is made to the server. Although limiting the Web Listener to a single process eliminates server overhead associated with starting new processes, this configuration will not respond well in a multi-user environment. To help improve performance, the Web Listener offers several different features. First, it allows the WebServer administrator to cache commonly accessed files in memory. The Web Listener provides logs that the administrator can use to determine which files are frequently accessed and thus, are candidates for caching. Additionally, the Web Listener supports the ability to perform memory mapping of files. With memory mapping of files, the Web Listener automatically maps files into memory addresses when they are accessed. By doing this mapping, the file is in effect loaded into memory, and as a result, connections accessing the same file do not have to perform duplicate disk reads. Furthermore, many OSs can pre-fetch the next segment of one of these memory-mapped files while the previous segment is being transmitted. This mechanism offers improved performance in single connection cases as well as multiple connection cases.

The Oracle Web Listener supports several security mechanisms. Foremost among these is authentication. The Web Listener supports two types of authentication: basic and digest. Both authentication types require users to supply user names and passwords in order to access documents. In the case of digest authentication, however, the user names and passwords are also encrypted before they are transmitted across the network. The encryption is performed using a cryptographic checksum to encode the password, and thus, it prevents attackers from obtaining the original username/password combination. In order to implement digest authentication, however, the browser must support it as well, and the Netscape browser does not support it. Basic authentication is independent of the browser because it is implemented only at the HTTP server.

The Web Listener also supports restricting access based on IP addresses. Access may be restricted to specific IP addresses or to entire IP subnets and networks. This security may also be implemented using host names rather than actual IP addresses. Authentication, especially digest authentication, offers a more secure environment than IP address restrictions as attackers may falsify IP addresses to gain access. Any one of the authentication and restriction methods may be applied to specific files in any combination. Finally, in Unix environments, i.e., Solaris 2.x and HP-UX, the Web Listener's access to files on the server may be limited by specifying a user and group ID for the Web Listener. Unix security mechanisms will then allow the Web Listener to access only those files to which the Web Listener has permission.

The Web Listener logs request information in a specified log file. First, it logs the information about the clients request. This information includes the action to be taken, the URL and the protocol version. It also logs which client is making the request, whether or not the request was successful

and the number of bytes transmitted as a result of the action taken. This information may be useful in terms of security and enhancing performance.



2.3.2.3 OWA. The Oracle Web Listener uses Version 1.1 of the CGI to execute programs that can dynamically build HTML documents. As already stated, the OWS includes the OWA, a CGI program that facilitates the execution of PL/SQL procedures that are specified as part of the URL. The OWA and PL/SQL are discussed in more detail in the following two sections.

2.3.2.3.1 OWA's Operation. The OWA operates in the following manner. First, the user submits a URL using the client's browser. The URL contains information that tells the Web Listener to spawn a Web agent process, and it specifies the name of a Web agent service that must be invoked by the agent. The Web agent service contains information such as which Oracle7 database to connect to and which username and password to use. The Web agent determines which service to use from information contained within the URL. After connecting to the database, the Web agent executes the PL/SQL stored procedure specified in the URL. Then, the PL/SQL procedure extracts some data from the Oracle7 database and generates an HTML document which it sends to standard output for receipt by the Web Listener. The Web Listener is then able to transmit the document to the browser via HTTP. Figure 2-4 summarizes this process.

Figure 2-4: Summary of Web Agent's Operation.

2.3.2.3.2 OWA's Use of CGI Environment Variables. As already stated, the submitted URL contains information used by the OWA. The OWA obtains this information via CGI environment variables as described in Paragraph 2.2. Table 2-2 summarizes the CGI environment variables used by the OWA when it parses a submitted URL.

Table 2-2: CGI Variables Used by the OWA to Parse URLs.

Variable	Contains
REQUEST_METHOD	Tells the OWA which method is being used to pass data. The OWA supports the GET and POST methods.
PATH_INFO	Tells the OWA the name of the PL/SQL procedure to invoke.
SCRIPT_NAME	Tells the OWA the service name it should use when logging into Oracle7.
QUERY_STRING	When using the GET method, tells the OWA which parameters should be passed to the PL/SQL procedure (recall that POST method parameters are passed via standard input).

In order to understand more clearly how the OWA uses these CGI variables, the following URL is presented as an example to show how a typical URL is parsed when using the GET method:

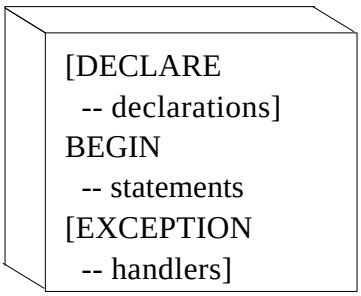
```
http://jdic4.gcc.sra.com:8887/
ows-bin/Web/owa/air_carrier_page.display?type=blue
```

In the previous URL, the substring, `http://jdic4.gcc.sra.com:8887`, tells the Web browser to connect to port 8887 of the host specified by jdic4.gcc.sra.com via the HTTP protocol. The Web Listener, which is listening on port 8887 of jdic4.gcc.sra.com, recognizes the `/ows-bin/Web/owa` substring as a request to spawn the OWA. The Web Listener then passes `/ows-bin/Web/owa` to the Agent as the environment variable SCRIPT_NAME. The OWA parses SCRIPT_NAME to extract the service name it should use to access the database which in this case is `Web`. The substring `air_carrier_page.display` is passed to the Web Listener as PATH_INFO. This environment variable specifies the PL/SQL procedure, in this case `air_carrier_page.display`, that the Agent should invoke on behalf of the client. When the OWA invokes the PL/SQL procedure, it uses Oracle's unpublished, proprietary CLI. The substring `type=blue` is passed to the Agent as QUERY_STRING. The agent parses QUERY_STRING and passes these parameters to the PL/SQL procedure that it invoked.

The previous process is only used with the GET method. With the POST method, the Web Listener passes parameters to standard input where the OWA reads them and passes them to the specified PL/SQL procedure. The important point is that in both cases the method chosen for passing parameters is transparent to the PL/SQL procedure, and thus, developers of stored procedures do not have to concern themselves with which method is being used. This fact is not true of the GUI which must specifically address the method for passing parameters. In addition to the CGI environment variables in Table 2-2, the OWA can make use of the environment variables in Table 2-1 in Paragraph 2.2.

2.3.2.3.3 OWA's Error Handling. The OWA handles two types of errors: application and system errors. Application errors are specific to PL/SQL procedures, and as a result, each procedure must include its own exception handling and associated output. This output is passed on to the OWA as

HTML which is in turn passed on to the browser by the Web Listener. As far as the OWA is concerned, if the PL/SQL procedure generated HTML output, then the procedure was successful. These types of errors are transparent to the OWA. System errors, on the other hand, are actually detected by the OWA. These errors occur when the OWA is unable to execute a specified PL/SQL procedure or when a PL/SQL procedure does not handle a particular exception, causing the exception to be propagated to the Agent as a system error. In this case, the HTTP server returns a standard HTML error document to the browser. Oracle's HTTP server also supports the development of custom HTML error pages.



2.3.2.3.4 PL/SQL. It has already been explained that the WebServer uses PL/SQL procedures to carry out database operations. PL/SQL is a block-structured language, meaning that PL/SQL programs are composed of logical blocks that each correspond to a problem to be solved. The basic form of a PL/SQL block is shown in Figure 2-5.

Figure 2-5: Structure of a PL/SQL Block.

The declarations portion is where PL/SQL allows the developer to declare variables and constants that are used in the executable portion. The BEGIN, or executable, portion is where the business rules and data I/O layers are implemented. The exception handling portion contains routines to deal with exceptions raised during execution. PL/SQL uses an object-based syntax similar to Ada, meaning PL/SQL supports some object-oriented concepts such as encapsulating data and methods into packages but does not support fully object-oriented constructs such as inheritance or polymorphism. There are two parts to a package: the specification and the body. Package specifications contain an object's interface which is essentially the names of procedures that may be called by other packages. Package bodies contain the actual code responsible for implementing the procedures contained in the specification and other procedures that can only be called from within the package. Packages are important because they allow code to be reused, thereby reducing development time.

The PL/SQL programming language includes standard programming features such as flow control statements and loops; variables constants and types; structured data and customized error handling. Additionally, PL/SQL supports almost all the SQL data manipulation commands, giving PL/SQL an extensive ability to access Oracle data. SQL is a non-procedural language that allows developers to issue instructions to DBMSs. DBMS vendors typically support slight variations of the SQL standard. These traits allow the PL/SQL to support the development of complex business rules.

More specifically, PL/SQL allows for the incorporation of transactional modifications and for the nullification of these transactions if any one part of the transaction fails.

Another salient point about PL/SQL stored procedures is they must reside in the Oracle7 database. This point has a couple of important ramifications. By residing within the Oracle7 database, PL/SQL stored procedures also run within the Oracle7 database. As a result, PL/SQL procedures are portable to any hardware and OS platform that support the Oracle7 database server which is widely supported. This portability implies that PL/SQL procedures can be moved from one platform to another without recompiling the procedure as long as the platforms involved support the Oracle7 database server. This feature is obviously advantageous, but there are also drawbacks associated with a dependency on the Oracle7 database. Recall that one of the design requirements is for the Web application development tools to support a modular architecture that separates system layers, allowing them to distribute processing throughout a network. If stored procedures reside in the database, then the business rules and data I/O layers are not separate from the DBMS layer. Consequently, it would be difficult to distribute these layers to different machines. Distribution could be achieved by using an Oracle7 database that merely acts as a storage and execution space for the procedures. The procedures could then access another Oracle7 database on a different server. This solution, however, is less than ideal and represents a workaround at best. This topic is discussed again in the next section. Another drawback to stored procedures residing in the database is that they execute within the database, thus using valuable database resources. In general, stored procedures do not lend themselves well to the modular concept described in the requirements section.

2.3.2.3.5 Database Access for Sybase and Informix. The PL/SQL applications which are written to access the Oracle7 DBMS can also access Sybase and Informix DBMSs through the use of Oracle's Transparent Gateway product, an additional product. This gateway allows applications to access Sybase and Informix data transparently using ODBC as the CLI. The gateway provides two levels of transparency: data access transparency and database location transparency. With data access transparency, applications that are designed to access Oracle7 using SQL data manipulation commands that Oracle7 understands do not have to be rewritten to use SQL that Sybase and Informix understand. Instead, the gateway takes care of any necessary conversions to the SQL statements. Additionally, it converts any retrieved Sybase or Informix data into a form compatible with Oracle7. With the second type of transparency, database location transparency, applications need not know where the Sybase or Informix database is located. This transparency is achieved by using synonyms within the Oracle7 database which point to database links to Sybase or Informix tables. Furthermore, the gateway does not have to reside on the same server as any of the databases involved. If the gateway is located remotely to the Oracle7 server, then they communicate with one another via SQL*Net. If the gateway is located remotely to either the Sybase database or the Informix database, then they communicate using Sybase's or Informix's networking equivalent of SQL*Net. Figure 2-6 shows this type of arrangement.

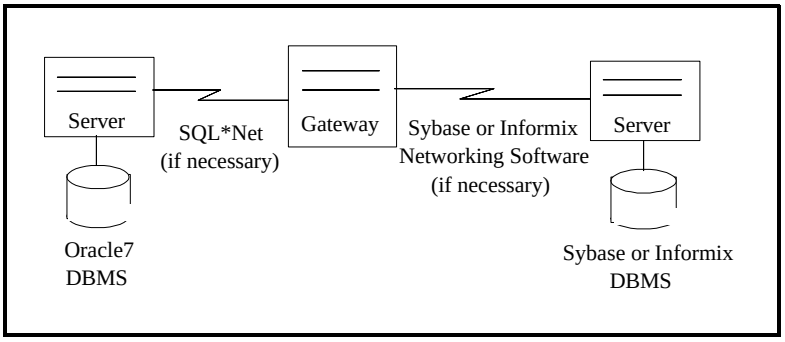


Figure 2-6: Oracle's Transparent Gateway.

Data access and database location transparency lead to another important feature: the ability to perform heterogeneous transactions. With heterogeneous transactions, data residing in Oracle, Sybase and Informix databases can be accessed with a single SQL statement, thus allowing for heterogeneous table joins and subselects.

Using an Oracle gateway in conjunction with PL/SQL stored procedures to access data in Sybase and Informix databases has a major drawback: the reliance on having an Oracle7 database. Even in situations where it is only necessary to access Sybase or Informix databases, it is necessary to have an Oracle instance running on the applications server. This instance will use valuable server resources such as Central Processing Unit (CPU) time, memory and disk storage. Furthermore, having an Oracle instance in this environment is really an unnecessary cost and complicates configuration management.

The Oracle transparent gateway operates within the following hardware and OS environments: HP9000 running HP-UX 9.04 or higher, Sun SPARC-based systems running Solaris 2.3 or higher, and IBM RS/6000s running AIX 3.2 or higher. The Oracle gateway is not supported on the Windows NT platform.

2.3.2.4 OWS Developer's Toolkit. The WebServer Developer's Toolkit includes several PL/SQL packages designed to facilitate the development of Web-based applications. Two of these packages are the HyperText Procedure (HTP) package and the HyperText Function (HTF) package. These two packages help to minimize the amount of HTML syntax a developer must know because they eliminate the need to hard code the exact syntax of the HTML in PL/SQL procedures. With the HTP and HTF packages, developers still need to have a working knowledge of HTML tags, but the actual building of an HTML tag is accomplished by the procedures and functions in the HTP and HTF packages. An HTP generates a line in an HTML document that contains an HTML tag corresponding to its name. For example, the htp.anchor procedure generates an anchor tag. An HTF returns the HTML tag that corresponds to its name. The Toolkit also contains OWA utilities, a set of procedures and functions which perform utilities ranging from printing a signature HTML page to easy formatting of Oracle tables into HTML.

2.3.2.5 OWS 2.0. This section highlights the more important attributes of OWS 2.0 that are not present in OWS 1.0. In addition to including a standard multi-threaded, single process HTTP server that can execute external programs via the CGI as described earlier, OWS 2.0 implements an asynchronous request broker called the WRB. The WRB is in effect an open API that handles requests and interfaces with WRB cartridges via a pool of processes called WRB Executable (WRBXs) engines. Figure 2-7 depicts this architecture.

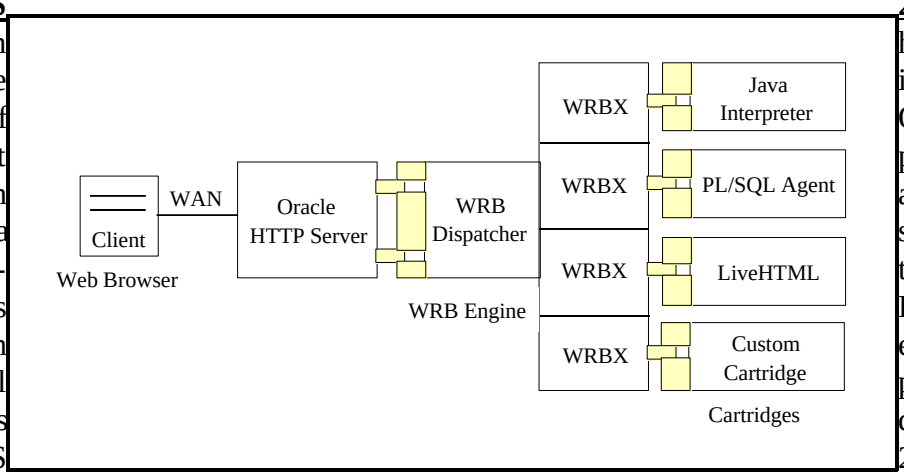


Figure 2-7: OWS 2.0 Architecture.

The WRB architecture should provide improved performance over the CGI which must spawn a process each time a request is made to execute a CGI program. Currently, the WRB includes LiveHTML, PL/SQL and Java cartridges. The LiveHTML cartridge provides a way to produce HTML pages that contain dynamic content. The PL/SQL cartridge allows Web applications to be developed using stored procedures. From a development standpoint, the PL/SQL cartridge operates in the same manner as the OWA in OWS 1.0. The major difference is that, in OWS 2.0, stored procedures are executed via WRBXs rather than via the CGI used by OWS 1.0. The Java cartridge is a Java interpreter that allows for the development of server-side Java. Business rules can be coded in Java and executed at the server. Additionally, Oracle has added some Java classes that allow the developer to connect to the Oracle database. These classes, however, do not enable Java to perform database updates and queries. Instead, to perform these operations, any developed Java code must execute stored PL/SQL procedures. Once again, there is a reliance on PL/SQL stored procedures, and the problems described in the OWS 1.0 sections still exist. The WRB allows developers to write

custom cartridges. By creating custom cartridges, developers can enable the WRB to execute external applications such as Common Object Request Broker Architecture (CORBA) or Object Linking and Embedding/Component Object Model (OLE/COM) based object-oriented applications written in C++ via the WRB. Oracle's migration towards an open API, to execute external programs, that does not have the performance limitations associated with CGI represents a trend in the Web marketplace; vendors are striving to have their individual APIs become the standard

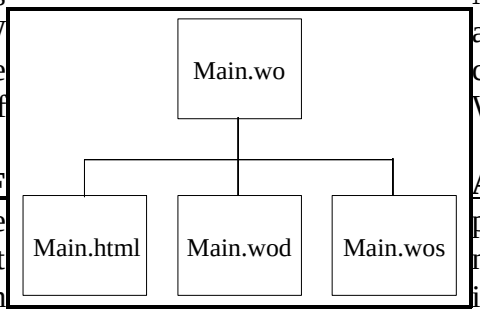
Another important area that Oracle has changed in OWS 2.0 is the level of security supported by the Web Listener. Recall, the 1.0 Web Listener supports basic and digest authentication. The 2.0 Web Listener adds SSL to its security. Of course, as is the case for basic and digest authentication, the Web browser must support SSL which the Netscape browser does support. The addition of SSL bolsters the WebServer's ability to provide for secure transmissions between the client and the server.

Oracle is striving to make the OWS more flexible when it arrives as the Version 3.0. An important additional capability that Oracle plans to add in WebServer 3.0 is the ability for cartridges to communicate with and pass data between one another. This ability will allow cartridges to make use of other cartridges such as an ODBC cartridge to access other vendors' DBMSs. Another important area Oracle plans to address in the Version 3.0 is the idea of a stateful server. The Versions 1.0 and 2.0 do not provide any functionality for keeping track of state at the server. In WebServer 3.0, customized cartridges using context handles may be used to achieve state at the server. The importance of this limitation is explained in detail in the prototype section. To improve interoperability, Oracle is working on providing WRB adaptors for HTTP servers other than Oracle's Web Listener.

2.3.3 WebObjects Enterprise

NeXT's solution for bringing database applications to the Web is WOE. WOE consists of the WOF and the Enterprise Objects Framework (EOF) and can make use of another NeXT product, Portable Distributed Objects (PDO), to distribute objects. The WOF provides developers with an environment in which to write Web-based applications, while the EOF provides database access classes that can be used by the WOF applications. WOE is currently available on the Solaris 2.x, HP-UX 9.x, and Windows NT OSs. There is, however, one part of the WOE that is not currently supported on the above OSs: the Enterprise Object Modeller (EO Modeller) used by the EOF. Currently, this product is available on NeXT's NEXTSTEP Mach OS. NeXT also has a beta version of a Web-based EO Modeller and plans to have a shipping Web version in June 1996. The Web-based version of the EO Modeller would in effect support Solaris 2.x, HP-UX 9.x, and Windows NT because of the hardware platform and OS independence associated witth the Web. NeXT also plans to develop a Windows NT version of the EO Modeler for September 1996. In addition to these supported platforms, WOE operates with any HTTP server that provides a CGI interface and with any Web browser. WOE also supports the NSAPI to help improve performance when using Netscape HTTP servers. Furthermore, WOE applications can leverage any security offered by HTTP servers. NeXT does not offer an HTTP server, nor is it NeXT's intention to develop an HTTP server. This component, therefore, is not discussed in this section as it is in the Oracle and Netscape sections. The following sections discuss the WOF and the EOF, as well as PDO, in more detail.

2.3.3.1 WOF. The WOF is NeXT's environment for building and deploying WWW applications. The WOF provides a scripting language called WebScript and objects that facilitate the creation of Web applications.



2.3.3.1.1 Creating WOF Applications. A typical WOF application consists of three parts: components, WebObjects classes, and an executable that manages the running application. If the application is completely written in script, a supplied default application, called DefaultApp, is used. If the application uses compiled code, a supplied C main is used and the compiled application is linked with the WOF framework library. In both cases, the program is responsible for receiving incoming requests and responding to them using components provided by the developer. WebObjects classes provide the infrastructure that programmers need to develop Web-based applications. A class defines variables and methods that are associated with objects created from that class. Objects are particular instances of classes, and methods are procedures that can be executed by objects. It is the components that specify the content, presentation and behavior of an application's pages. The components of a WebObjects application will now be discussed in more detail. Components are comprised of three files: an HTML template, a script file, and a declarations file. The HTML template, as indicated earlier in the discussion on HTML, contains information in the form of hypertext markup elements that define how the generated HTML page should look. The script file implements application behavior that is specific to the component being developed. In general, script files declare variables for managing the content of dynamically generated HTML pages and the actions that define responses to user requests. The declarations file serves as a mapping between the HTML template and the variables and actions defined in the script file. These three files are organized into a component directory as shown in Figure 2-8.

Figure 2-8: WOF Component Structure.

In Figure 2-8, the particular application being developed has a component named Main.wo. The HTML template, script file and declarations file are also named Main, but they have different extensions. The HTML template has the extension .html, while the script and declarations file have the extensions .wos and .wod respectively. An important point about components is that they are reusable, meaning that more than one WebObjects application can use a particular component.

In order to better understand how the HTML template and the script and declarations files interoperate, the following example of how to write an application that takes user input and then generates a dynamic HTML page based on that input is presented. More specifically, the application allows the user to input a name, and then the application displays a second page that says "Hello name" after the user has submitted the first page via a submit button. With WebObjects, writing an application requires the creation of a component for each page in the application. In this example, therefore, there are two components. The DefaultApp program automatically looks for a Main component, Main.wo, as the first page if not directed to load a particular page. A typical URL for invoking a WebObjects application is:

```
http://jdic4/cgi-bin/WebObjects/AddCarrier
```

where `jdic4` is the hostname, `cgi-bin` refers to the HTTP server's CGI directory and WebObjects/AddCarrier fully specifies the directory containing the main component of the application.

Now, the first step of the process is to write an HTML template for the main page which may look something like this template:

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
<HEAD>
<BODY>
<FORM>
What's your name
<P>
<WebOBJECT NAME = "NAME_FIELD"></WebOBJECT>
<WebOBJECT NAME = "SUBMIT_BUTTON"></WebOBJECT>
</P>
</BODY>
</HTML>
```

With the exception of the two `WebOBJECT` elements, every tag is a standard HTML tag. The two `WebOBJECT` tags are used by the WOF as a way of introducing dynamic HTML to the page when it is returned to the HTTP server by the application.

The next step in the process is to develop the declarations file which specifies the kind of objects that perform the substitutions of the dynamically generated HTML. For this example, the declarations file would look like this:

```
NAME_FIELD: WOTextField (value = nameString)
SUBMIT_BUTTON: WOSubmitButton (action = sayHello)
```

Note that each line is a declaration that creates an object, known as a WODynamicElement, that represents the corresponding `WebOBJECT` element contained in the HTML template. This declaration serves to map the WODynamicElement objects to variables and actions contained in the script file.

The script file for this example would have the following content:

```
id nameString
- sayHello
{     id nextPage;
      nextPage = [WOApp pageWithName:@"Hello"];
      [nextPage setNameString:nameString];
      return nextPage;
}
```

It is the script file that contains the code that is considered the business rules of an application.

Working together, these three files define the action that must take place when a user submits a name: return a second page with the "Hello name" greeting. In order to accomplish this action, the application must first store the name entered by the user so it can use it to generate the second page dynamically. This piece of the process has already been addressed. The line `NAME_FIELD: WOTextField (value = nameString)`tells the WOF how to store the name by associating the NAME_FIELD element with the nameString variable declared in the script file. In short, this declaration tells the WOF to create a WOTextField object that generates HTML for the NAME_FIELD element. Furthermore, it specifies that the value attribute of the WOTextField Object will be set to the value of the variable nameString which represents the user's input.

Now, all that is left to do is to return the second page which says "Hello name". This action has already been defined in the line `SUBMIT_BUTTON: WOSubmitButton (action = sayHello)`. This declaration associates the SUBMIT_BUTTON element with the sayHello method defined in the script file. As was the case with storing the name, this declaration tells WOF to create a WOSubmitButton object that generates HTML for the SUBMIT_BUTTON element. Furthermore, it specifies that the action assigned to this object, sayHello, will be invoked when the user submits the form. The sayHello method is responsible for invoking a second component, the Hello component. As the first component represents the user input page, this second component defines the "Hello name" page that is returned. This component, therefore, also has an HTML template, a declarations file and a script file. The HTML template would look like this:

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>
<BODY>
Hello <WebOBJECT NAME = "NAME_STRING"></WebOBJECT>!
<!BODY>
</HTML>
```

The declarations file would have the following line:

```
NAME_STRING: WOString {value = nameString};
```

The script file would have the following entry:

```
id nameString;
```

The sayHello method defined in the Main component sets the Hello component's nameString variable to the value entered by the user. The sayHello method performs this action using what is

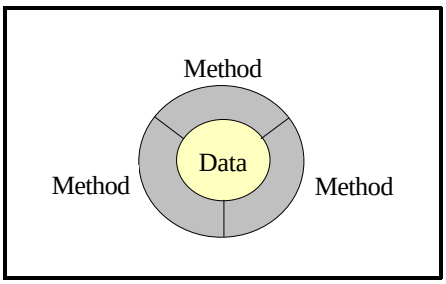
referred to as an accessor method. WebObjects incorporates two types of accessor methods: set and get. The set accessor method allows a component to set the value of a variable in another component's script file. The get accessor method allows a component to return the value of a variable in another component's script file. Now, the Hello component has access to the information entered by the user on the previous page, and it can return a dynamic HTML page that says "Hello name".

There are some important points to make about the previous example. First, there were several times when an object was created from a class such as WODynamicElement, WOTextField, WOSubmitButton, and WOString. These classes are part of the WOF and can be accessed by WebObjects applications via the DefaultApp program. The DefaultApp program is an Objective C program, discussed in more detail in subsequent sections, that links in the WOF classes and then serves as an access mechanism to these classes. In other words, developers using the WebScript language, the WOF's scripting language, can instantiate objects of the WOF classes without having to link in these classes. The DefaultApp program is in effect an interpreter for the WebScript programming language. This point leads to another important fact; WebScript is an interpreted language. Being an interpreted language, WebScript programs are not compiled and do not need to link in classes. Furthermore, applications written in WebScript are hardware platform and OS independent. It must be understood, however, that the program responsible for interpreting the WebScript program, in this case the DefaultApp program, is a compiled program, and as a result, it is not platform and OS independent. Another aspect of interpreted programs is they are generally slower than compiled programs. This fact is true because each time an interpreted program is run, it must be interpreted. A compiled program, on the other hand, is simply executed.

In addition to being an interpreted language, WebScript is an object-based language that is a subset of the Objective C programming language. As an object-based language, WebScript support the development of functionality in the form of objects that are reusable. There are, however, some drawbacks to using WebScript. First, although the WOF classes that support database access are available, a program written entirely in WebScript must store all retrieved data in enterprise object generic (EO generic) records which do not allow the developer to fuse data and business policy together in objects. The importance of this limitation will become more apparent during the discussion of the EOF. The second drawback is the inability of WebScript to create new classes. Because WebScript is not fully object-oriented, it can only make use of classes that are delivered with WOR, or that are created using the compiled Objective C programming language. This limitation can, in many cases, hinder the development of complex business rules..

Due to the limitations of the WebScript language, it may prove necessary to develop business rules in a more complete programming language. In keeping with this idea, the WOF supports the development of business rules in the Objective C programming language. WebScript provides two ways for developers to make use of Objective C. With the first way, developers can use the WOF classes, WOApplication, and WOWebScriptComponentController, to write methods that can access custom-developed Objective C methods. The second way is for developers merely to call Objective C classes from within the script file using WebScript. The second approach is more desirable because it allows the developer to continue to make use of the WODynamicElement classes that are a part of the WOF. Objective C is implemented as a set of extensions to the C programming language which provide a complete capability for object-oriented programming. The benefits of this

implementation are two-fold. First, the developer has the ability to implement business rules in an object-oriented manner when desirable, and second, the developer still can choose to implement business rules with more procedural programming techniques as needed.



As an object-oriented extension to C, Objective C is similar to C++, but Objective C is much smaller and therefore more easily learned than C++. Additionally, Objective C uses a dynamic object model, while C++ uses a static object model. An object model specifies a language's core facilities for using objects. A dynamic object model can more easily adapt to changes than a static model. For example, applications based on dynamic models can be modified to use new objects and new methods without recoding, recompiling or restarting which is not the case with applications based on static models. A drawback of dynamic object models is they tend to be more error prone than static models. Objective C provides facilities for performing messaging and typing statically rather than dynamically to improve reliability.

From the standpoint of comparing Objective C to WebScript, Objective C does not suffer from WebScript's lack of string manipulation capabilities. Another important characteristic of Objective C is that it is a compiled language. Consequently, it has the benefit of being generally faster than WebScript applications, but it does not enjoy the platform and OS independence that accompanies WebScript. Compiled programs make configuration management and process distribution more complex because each OS requires its own compiled versions of the applications. If changes are made to an application, the application must be recompiled for each particular OS in the environment and redistributed. Consequently, there is a trade off in using a compiled language like Objective C as opposed to an interpreted language like WebScript. In this case, however, Objective C is the clear choice because of its ability to implement more complex business rules than WebScript.

2.3.3.2 EOF. The EOF is an important part of NeXT's solution for deploying applications to the Web because the WOF is able to use the EOF to update and query the data in relational databases. The EOF does more than just provide access to enterprise data because it is able to represent relational database elements as objects that can be used by WebObjects applications. By representing database elements as objects, it is possible to fuse business rules, known as methods, with the enterprise data, referred to as instance variables, as shown in Figure 2-9.

Figure 2-9: An Enterprise Object.

EOF accomplishes this representation by generating objects dynamically at run time from developer supplied definitions of data and methods. To understand the implication of coupling business data with business logic, the following example is given. A utility company may use customer objects in its billing system. Within each customer object, is information such as money owed to the company. Along with this data may be a formula or business rule for calculating the amount of money each customer owes. If the business rule for calculating the customer's debt changes, then the customer's debt will change accordingly the next time the customer object is used. An additional implication of representing database elements and business rules as objects is reusability. If other applications also need to use customer objects, then they can, ensuring that the same method for calculating a customer's debt is used in different applications. Reusing objects can also decrease development time and prove to be more reliable than using new objects because the reused objects have already been tested and used.

In order to represent database elements as objects, the relational data model must be mapped to an object model. NeXT allows the EO Modeler to automate the mapping process. This tool allows developers to define the mappings between business objects classes and the physical database. The mapping process is based on an Entity/Relationship (E/R) approach that is independent of a particular vendor's database. This approach is one of the best known and most commonly used database modeling methods. EOF uses E/R in the following manner. Business object classes map to abstract entities that correspond to table or views within the database. Entities are comprised of attributes and relationships. Attributes correspond to columns while relationships correspond to joins between primary and foreign keys. Note that business object classes are not limited to mapping to a single table. Instead, a business object class may map to multiple tables and have properties that do not correspond to database columns. After this mapping is established, EOF is able to perform the automatic translation of database records into objects at run-time, and developers can create custom Objective C methods that belong to these objects. Now, the business data is linked to the business rules that use and modify that data. Once these Objective C classes are created, they can be called from either WebScript or other Objective C code.

It is important to understand that fusing business rules with data in objects does not imply that the business logic and database I/O layers cannot be separated from the DBMS layer. Instead, EOF insulates the business logic and database I/O layers from the DBMS layer. By separating these layers, applications can access different databases without recoding business logic or database I/O code. This insulation also allows a single application to use data from multiple data sources such as relational databases from a single vendor or from multiple vendors. Furthermore, if the database schema changes, then the developer merely has to change the mapping instead of the business logic or database I/O code. The objects are changed appropriately when they are generated at run-time. EOF achieves separation between the RDBMS layers and the business logic and database I/O layers by abstracting the interfaces to relational database through a database-independent API and then by using adaptors that provide database specific implementations of the general API. Table 2-3 contains information on the database adaptors that NeXT bundles with EOF.

Table 2-3: Database Adaptors Bundled with EOF.

Adaptor	DBMS	Availability
Informix	Informix	Now with Beta EOF 1.2 and Alpha EOF 2.0
ODBC	Any DBMS that has ODBC drivers for Windows NT, including Microsoft SQL Server and DB2	With Beta EOF 2.0 in Q2 of 1996.
Oracle	Oracle	Now with Beta EOF 1.2 and Alpha EOF 2.0
Sybase	Sybase	Now with Beta EOF 1.2 and Alpha EOF 2.0

The Informix, Oracle and Sybase adaptors use the ESQL, OCI and DB-Lib/CT-Lib CLIs respectively. Additionally, EOF, through the use of Objective C to code business rules, supports the ability to perform transactional modifications as defined in Paragraph 2.1.1.1. EOF also supports the ability to rollback transactional modifications that have only partially been completed.

2.3.3.3 PDO. Another part of the NeXT Web application development environment is PDO which consists of the PDO object model itself and of a set of classes that can be linked in by Objective C at compile time. The PDO object model uses Objective C's dynamic object model and consequently, gains the high degree of flexibility offered by Objective C's object model. PDO's primary function is to enable messages to be delivered between objects, allowing objects to be distributed in the following ways:

- Objects can be distributed among disparate processes running on the same server
- Objects can be distributed among servers residing on the same LAN
- Objects can be distributed among servers connected via a WAN.

This flexibility allows developers to distribute objects, i.e., processing, to the network resources that are best suited to accommodate specific objects. Figure 2-10 summarizes each step of the process by which a developer distributes an object.

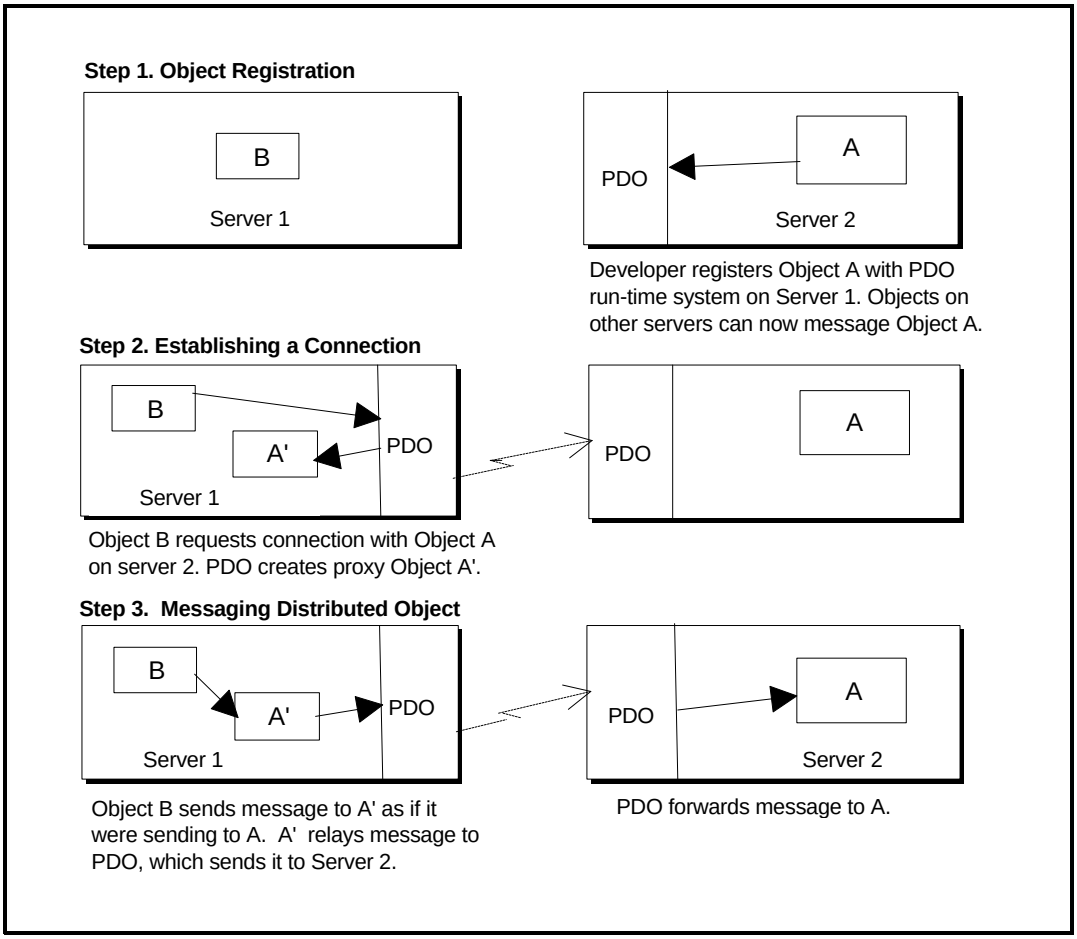


Figure 2-10: Distributing Objects with PDO.

PDO also brings increased interoperability to a Web-based application in terms of hardware and software interoperability. PDO allows objects residing on disparate hardware and OS platforms to interoperate. Table 2-4 contains the hardware and OS platforms supported by PDO.

Table 2-4: PDO Hardware and OS Platform Support.

Hardware Platform	OSs Supported
Intel*	Windows NT*, Windows 95 and Solaris
SPARC*	Solaris 2.x*, Mach
PA-RISC*	HP-UX 9.x*, Mach
NeXT	Mach
Alpha	OSF/1
PowerPC	Solaris

The * indicates server hardware and OS platform combinations that are pertinent to the DII COE.

Another type of interoperability that PDO offers is interoperability with other object models. More specifically, PDO interoperates with OLE/COM, thus allowing PDO developed applications to use OLE-based applications. NeXT also has plans to support CORBA in the future but has not announced a date. OLE/COM and CORBA are two of the most prevalent object models in use today.

2.3.4 Netscape

Netscape offers a wide range of products which fall into the realm of Web technology. Two of these products, the Netscape Commerce Server 1.2 and LiveWire Beta 2, are particularly important in this evaluation because they work in conjunction to provide database access to a Web client. At the end of April, during the latter part of SRA's evaluation, Netscape informed SRA that the 1.0 production version of LiveWire which is currently in Beta 4 will not be designed to operate with the Commerce Server. Instead, it will operate with the Enterprise Server, a new server being developed by Netscape that is currently in Beta 3. The Enterprise Server offers more functionality than the Commerce Server, and both servers will be discussed in more detail. Netscape has informed SRA that LiveWire Beta 4 adds the following functionality to LiveWire Beta 2:

- Support for server-side image map handling, i.e., the ability to get X and Y coordinates clicked by a user
- Support for handling responses with multiple options selected on an HTML form select list
- Additional capabilities for maintaining client state in URLs
- An implementation of the Site Manager, discussed later, for Solaris
- Some sample applications that demonstrate server-side file I/O.

It is SRA's opinion that for this particular evaluation, this increased functionality would not greatly affect the recommendations and conclusions presented in this paper.

An additional point about the Netscape product suite that must be made is it is rapidly changing, and the accompanying documentation is not keeping pace with the changes in the products. Furthermore, much of the documentation made available to SRA by Netscape is marketing literature that does not include detailed information about how a product works.

2.3.4.1 Netscape Commerce Server. As previously mentioned, HTTP servers are not a focus of this paper, but they are particularly important in terms of security and performance of the overall system. The Netscape Commerce Server is designed to allow for secure electronic commerce and communications on the Internet and other TCP/IP-based networks such as the SIPRNET. All the server's communications are based on open standards. It uses HTML to publish documents that are transmitted via HTTP. The Commerce Server supports HTTP Version 1.0 which ensures that the

server is compatible with any HTTP-compatible clients and servers. Additionally, like the Oracle Web Listener and other HTTP servers, it follows the request/response model.

2.3.4.1.1 Performance and Security Features. The Commerce Server runs as a multi-process, single-threaded engine. In fact, the Commerce Server allows the administrator to specify a minimum number, or pool, of processes that are always available to handle HTTP requests. By having this pool of processes already spawned and constantly available, the Commerce Server avoids the additional system overhead associated with creating processes to service multiple requests and then deleting those processes after fulfilling the request. Additionally, the Commerce Server allows the administrator to specify the maximum number of processes that may be available at any given time. As the HTTP server load increases, the Commerce Server uses a dynamic process management algorithm to increase the number of processes within the limits specified by the administrator. As the load decreases, the Commerce Server shuts down the newly spawned processes until the load is low enough to be handled again by the original pool of processes. It is important to point out that a relationship exists between the number of available processes and the amount of system resources available for other applications or database instances that may reside on the same server as the HTTP server. If the initial pool of processes is too high or the Commerce Server is allowed to spawn too many processes to respond to higher loads, the performance of other processes, e.g., application and database, may be affected to an extent that makes them in effect unusable. Consequently, it is important for the administrator to fine tune this number so that the best possible performance can be obtained from the server.

The Commerce Server supports two different APIs for executing external programs: the CGI and the NSAPI. Netscape implements the standard CGI as discussed in the Oracle analysis section. The NSAPI is an extension, consisting of functions and header files, that allows programmers to customize and augment the functionality of Netscape HTTP servers.

As far as security, the Commerce Server implements Version 2.0 of the SSL described in Paragraph 2.3.1.4. One aspect that is missing from this implementation is client authentication using digital signatures. The Commerce Server only implements server authentication.

The commerce server is available on a wide range of hardware and OS platforms. Table 2-5 contains the platforms that have the most pertinence to GCCS.

Table 2-5: Commerce Server Platform Support.

Hardware Platform	OS Supported
SPARC	Solaris 2.3 and 2.4
PA-RISC	HP-UX 9.03, 9.04, and 10.0
Intel	Windows NT 3.5

2.3.4.2 Netscape Enterprise Server. The Netscape Enterprise server adds to the capabilities of the Commerce Server. A major addition to the Enterprise server is its support for server-side Java

programs. Much like the Commerce Server supports the development of server-side JavaScript, the Enterprise Server supports the development of server-side Java programs that may be called using the NSAPI instead of the less performant CGI, although the CGI is still available to developers. Additionally, in the area of performance, the Enterprise Server is multi-threaded, meaning that one process may serve multiple users simultaneously. Additionally, unlike the Commerce Server, the Enterprise Server can take advantage of a server's symmetric multi-processors. For these reasons, the Enterprise Server should be more performant than the Commerce Server, especially in multi-user, multi-processor environments. For security, the Enterprise Server adds client authentication, advanced access control which allows the administrator to limit access to specific files and directories and support for SSL 3.0, an emerging SSL specification that is still being developed.

The production version of the Enterprise Server is targeted for the end of June 1996 on the Solaris, HP-UX, Windows NT, and IRIX OSs.

2.3.4.3 Netscape LiveWire. Netscape's LiveWire product is an on-line development environment for Web authoring and client-server application development. LiveWire consists of three major components: the Site Manager, the LiveWire compiler and server extension, and Netscape Navigator 2.0 Gold. The Beta 2 Solaris version provided by Netscape for the evaluation did not include Site Manager, but this component is discussed. LiveWire will be included with the Enterprise server and will not be required to be purchased separately. Navigator 2.0 Gold was not evaluated and is not discussed as it provides an environment for client-side HTML and JavaScript development, and the evaluation focuses on the use of Java for the GUI. LiveWire will be available for a variety of platforms. Table 2-6 includes the ones that are most pertinent to GCCS.

Table 2-6: LiveWire Platform Support.

Hardware Platform	OS Supported
SPARC	Solaris 2.4 and 2.5
PA-RISC	HP 9.x and 10.x
Intel	Windows NT 3.51

The production version of LiveWire is targeted for release at the end of July 1996 for the Solaris, HP-UX and IRIX OSs. For Windows NT, the target date is the end of June 1996.

2.3.4.2.1 LiveWire Site Manager. Site Manager is a graphical interface for LiveWire application developers and administrators of Web sites which supports several features that facilitate the creation and maintenance of Web sites. First, Site Manager offers a set of 24 templates for automatically creating new Web sites. These templates guide administrators through each step of the Web site creation process. Site Manager also supports a visual drag-and-drop interface for managing Web sites. This interface allows site administrators to move directories and files in the directory tree. Additionally, when moving or renaming files, Site Manager automatically updates the HyperText links that point to the relocated files. If Site Manager lists a particular link as invalid, then one of two cases has most likely occurred; the link does not exist, or there is a case problem in specifying the link. For the latter occurrence, Site Manager has a tool which automatically repairs case sense problems by comparing the links in a site to the actual file names in the directory tree and then by changing the HyperText links to match actual file names.

2.3.4.2.2 LiveWire Compiler and Server Extension. The LiveWire compiler and server extension are the heart of the LiveWire product. The compiler enables programmers to create LiveWire applications that are HTML files with embedded JavaScript code. The server extension consists of an object framework that gives developers capabilities to create database applications and of an application manager that allows programmers to add, modify, start, stop, and delete LiveWire Applications

2.3.4.2.3 LiveWire Applications. LiveWire uses JavaScript to allow programmers to increase client-side functionality and to build stand-alone server-side programs. When building client-side functionality, the LiveWire application typically uses HTML in conjunction with embedded JavaScript as in the following example:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
      function square(I) {
      document.write ("The call passed ",I," to the
                  function","&lt;BR&gt;")
      Return I * I }

</SCRIPT>
</HEAD>
</HTML>
```

In the previous example, all the code between `<SCRIPT Language="JavaScript">` and `</SCRIPT>` is JavaScript, and the rest of the code is HTML tags. This example contains a function that squares a number and then returns the new value.

Increasing client-side functionality using JavaScript was not the focus of this paper. Instead, SRA concentrated on using JavaScript to create server-side stand alone programs that implement business rules and data I/O functionality. Netscape's JavaScript implementation is in effect an object framework with the following pre-defined objects:

- Request: Contains data specific to the current transactions
- Client: Contains data specific to an individual client
- Project: Contains data for an entire application, allowing multiple clients to share data
- Server: Contains global data for the entire server, allowing multiple application to share data.

An important point to make is that the Client object allows some state information to be kept, but it does not provide state management for cursors which is desirable for the Geoloc Search functionality.

In addition to these objects, LiveWire includes a database connectivity library which allows developers to create server-side programs that can access a database. This library consists of JavaScript methods that allow developers to connect to databases, query a database for data, modify data, and manage transactions. In LiveWire, the scope of a transaction is limited to the current HTML page in an application, and within this scope, transactions may be committed or rolled back. Table 2-7 summarizes the database connectivity provided by LiveWire's database connectivity library.[1]

Table 2-7: LiveWire's Database Connectivity.

Database	CLI Used
Oracle	OCI
Sybase	DB-Lib/CT-Lib
Informix	ESQL
MS SQL/Server on NT	ODBC

2.3.4.2.4 JavaScript. As indicated, LiveWire uses the JavaScript programming language to allow developer's to create applications. JavaScript is an object-based scripting language that resembles Java but owes its origin to dynamically typed languages such as HyperTalk and dBASE. JavaScript also has similarities to the C language in that its language syntax is identical to C in most cases.

[1]LiveWire supports connectivity to Illustra databases using the Illustra CLI

Furthermore, JavaScript uses many of the standard control constructs and data types used by C such as “if”, “else”, “float”, and “int”.

As a scripting language, a primary goal of JavaScript is to simplify the complexity of programming. Additionally, another JavaScript goal of almost equal importance is to provide an easy to learn language for HTML developers to increase the capabilities of their HTML pages. SRA did not use JavaScript in this capacity. Instead, SRA actually coded server-side applications with JavaScript. The source file for a server-side LiveWire application is an HTML file that consists almost entirely of JavaScript. The only HTML present is an HTML <server> tag that informs the Commerce Server that this program runs on the server, not on the client. This source file is compiled into a .web file by the JavaScript compiler. The .web file is essentially a binary file containing platform independent bytecode. When a URL is submitted that points to a LiveWire application, Netscape’s HTTP server calls LiveWire’s run-time library via the NSAPI. The LiveWire run-time library then interprets the JavaScript bytecode, and the application actually runs within the address space of the HTTP server. Using the NSAPI to execute the LiveWire application should be more efficient than using the CGI

Another point is that JavaScript’s object references are checked at run-time, making it a dynamically binding language. Additionally, because the .web file is platform independent, it can be moved to and executed on any platform that is supported by the Netscape HTTP server and LiveWire without having to recompile it from the source HTML file. LiveWire is necessary because as already explained it provides the run-time libraries that interpret the JavaScript bytecode. This platform independence ameliorates the problem of configuration management.

2.3.5 Java

One of the goals of this evaluation is to determine the feasibility of developing GCCS applications using Web-based technology. An important aspect of GCCS applications is their GUI, and consequently, it is important to consider the development of a GUI as part of this evaluation. In particular, SRA evaluated the Java programming language as a means for developing a GUI. Java is an object-oriented programming language developed by Sun Microsystems to solve a number of problems with modern programming practices. The Java programming language developed from a research project to develop software for consumer electronic devices. Java is not an industry standard, but it is quickly gaining acceptance in the marketplace. Java can best be described by examining the goals that were used to design the language: to develop a simple, dynamic, object-oriented, distributed, robust, secure, architecturally neutral, portable, interpreted, high-performance, and multi-threaded programming language. The following sections examine each of these goals in greater detail.

2.3.5.1 Simple. Java is designed to allow developers to create applications without extensive training and to leverage current standard software development practices and procedures. Java is modeled after C and C++ programming languages, ensuring the existence of an extensive base of developers familiar with the language structure, syntax and development experience. Java, however, supports additional features which reduce the amount of code developer’s must write and, therefore, lead to simpler applications and less development time. Minimizing the physical size of a Java applet is especially important because the applet must be downloaded from the server to the client.

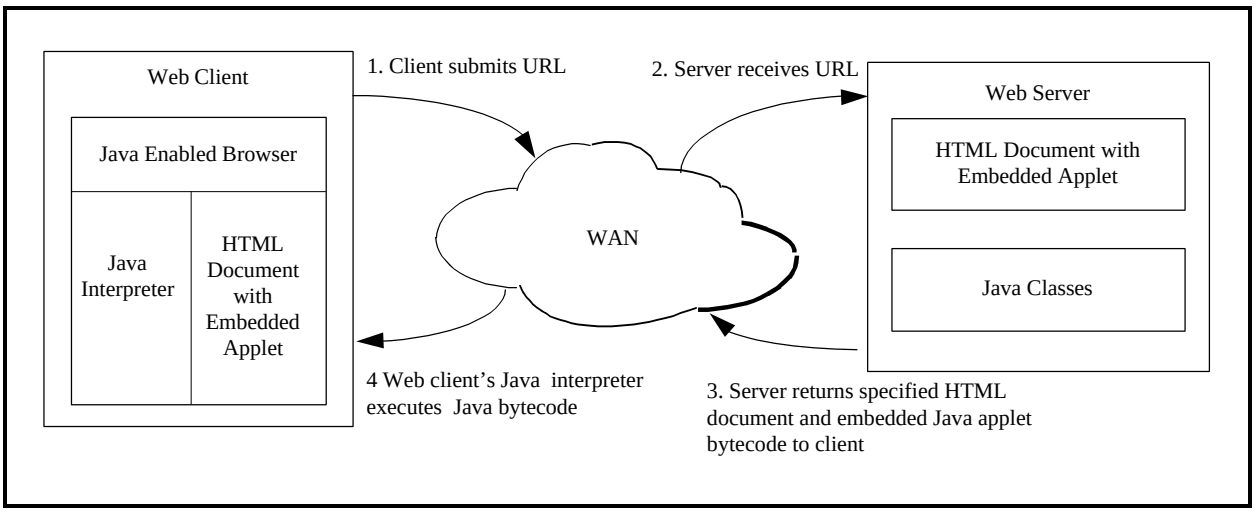
Obviously, the smaller the applet, the faster it can be downloaded. Two of these features are automatic memory management to free up memory resources when they are no longer in use and improved exception handling. Memory management is one of the most error prone aspects of custom developed software and exception handling can substantially increase the amount of code needed to create an application.

2.3.5.2 Object-Oriented. The concept of an object-oriented environment has been mentioned in the discussions on WebScript and Objective C, but a continued discussion of object-oriented concepts is appropriate. Object-oriented concepts focus design on the data, or objects, and on the interfaces, or methods, that can access that data. Java, being modeled after C++, contains most of the object-oriented facilities of C++ such as the ability to define classes with inheritance and a mechanism that allows member function calls to depend on the actual type of an object. The Java language, however, also eliminates several esoteric and error prone C/C++ features such as operator overloading, multiple class inheritance, templates, the "goto" control statement, global variables and the C language preprocessor. Furthermore, all functions must be implemented as methods, and all existing functions must be associated with a class. Finally, data variables or objects must be associated with a class, meaning Java does not support global variables. The object-oriented nature of Java leads to the creation of applications that are modular and reusable.

2.3.5.3 Distributed. The basic Java library provides an extensive collection of classes and methods for supporting Internet protocols such as TCP/IP and FTP[2]. Additionally, Java can access objects across the Internet using URLs. The following example shows how URL communications can be implemented with a minimum amount of code.

```
String  url = "http://www.site.com/homepage.html";
URL     theUrl;

try
      {
      theUrl = new URL(url);
       }
catch ( MalformedURLException urlException)
      {
      System.out.println("Bad URL: [" + url + "]");
       }

String            lineOResults;
DataInputStream   resultStream;
URLConnection     serverConnection;

try
      {
      serverConnection = serverUrl.openConnection();
      serverConnection.connect();

      resultStream = new DataInputStream(
                              new BufferedInputStream(
                              serverConnection.getInputStream()
                                                      )
```

[2]FTP does not work in the current Java release, and this fact is a known Java error. For a list of known Java errors refer to Appendix D.




```
                                        );

        while ((lineOResults = resultStream.readLine()) != null)
              {
              System.out.println(lineOResults);
              }
        }
catch (Exception ioException)
        {
        System.out.println("Error occurred while processing");
        }
```

An important point about this example is that all the used classes are basic Java language classes. Secondly, the applet is controlling the communication, not the browser. Once the URL is accessed, the information to which the URL points is returned to the applet as an uninterrupted stream of data which the applet processes in a manner similar to file I/O.

Java applets can also have the browser invoke a URL. When a URL is invoked in this manner, the applet terminates and is replaced with the document to which the invoked URL pointed. An example of invoking a URL in this manner is:

```
String url = "http://www.site.com/homepage.html";
getAppletContext().showDocument(url);
```

Java is distributed in another way as well; when a client invokes a URL that points to an HTML file with an embedded Java Applet this applet is downloaded to the client machine where it runs in the web browser's memory address space. Figure 2-11 summarizes this process.

Figure 2-11: Downloading a Java Applet.

When changes must be made to an applet, these changes only have to be made at the server level. The next time the client invokes the URL that points to the applet, the new applet is downloaded to the client. This capability clearly eases the burden of configuration management for clients.

2.3.5.4 Robust. Java is designed to develop applications that are reliable. This characteristic is achieved by eliminating features that can be misused or cause errors that are difficult to identify such as:

- € Pointer arithmetic and goto statements
- € A preprocessor
- € Pointers that manipulate memory as untyped objects
- € The necessity to free allocated memory dynamically.

The Java language does not just omit features to achieve robustness; it also adds features to achieve this goal. Java is a strongly typed language. For example, the compiler verifies all casting. In the C programming language, the following is valid:

```
int pointer;                        // a pointer
char buffer[5];                     // a memory buffer
*((char*)pointer) = buffer;         // point to the buffer
```

Later on in the program, the line `strcpy("This is too long for the buffer", buffer);` could appear and corrupt memory by overwriting memory that is not allocated to the `buffer` variable. The Java compiler, on the other hand, generates an error indicating an invalid cast operation has occurred. Java also requires that pointers point to objects that are of the same type as the pointer. This requirement eliminates the possibility of overwriting memory and corrupting data. An additional example is Java's support of true arrays. True arrays enable subscript checking and prevent access to memory which has not been allocated or is currently being used by other objects.

2.3.5.5 Secure. As already stated, the Java language and run-time environment is designed to be used in distributed environments. As a result, security is an important aspect of Java because Java could be misused to harm network resources and gain access to protected information. Java protects against this threat by means of a series of interlocking defenses that form an imposing defense against misuse and attack. Java provides security mechanisms at four different levels in its architecture. First, the compiler is designed to be defensive, meaning that it catches errors before they can occur and obeys the following safety rules:

- € Do not forge pointers
- € Do not violate access restrictions,
- € Do not access objects as a type that is different than the object's type
- € Do not call methods with inappropriate argument values or types
- € Do not overflow the stack.

Second, all bytecodes executed by the interpreter are screened to ensure they obey these safety rules. Third, the class loader, a portion of the interpreter that loads classes into memory, ensures that classes do not violate name spaces or access restrictions when they are loaded. Fourth, API specific security prevents applications from engaging in destructive activity by controlling the use of a set of well-defined methods that are allowed to be called by any given class and could be misused. This well-defined set of methods consist of file I/O, network I/O and methods that allow

one thread to access, control and manipulate other threads. For local file and network file access, the use of these methods may be restricted in the following ways.

- Unrestricted access which allows applets to do anything. Applets can read from and write to the local disk to which they are downloaded as well as the server from which they originated.

- Firewall access which allows applets that originate from within a firewall to read from and to write to the local disk to which they are downloaded as well as the server from which they originated if the server is within the same firewall as the client. This type of access may also apply to an Intranet environment like the SIPRNET.

- Source access which allows applets to read from and to write to the server from which they originated as well as to modify other applets residing on that server.

- No access in that applets cannot access both the client's disks and the server's disks. Applets are only allowed to run within the address space of the web browser.

It must be understood that the level of restrictions on the applet is implemented at run-time by the interpreter. As a result, when downloading applets to client machines for execution, the browser's interpreter determines the restrictions that are placed on the applet. These restrictions, therefore, may vary for various browser versions. Furthermore, some web browsers allow users to specify additional restrictions to some degree. For example, Netscape's web browser allows users to disable the Java interpreter completely to prevent applets from running on the client.

2.3.5.6 Architecturally Neutral. In order to help achieve the goal of being distributed, Java is designed to support the wide variety of hardware and OS platforms that may be found in a networked environment. In particular, to ensure a Java application can execute anywhere on the network, the Java compiler generates an object file consisting of bytecode instead of machine specific object code. Bytecode's design makes it easy to interpret on any machine. It can also be easily translated into native machine code by the Java interpreter at run-time, allowing any network processor with a Java interpreter to execute Java programs.

2.3.5.7 Portable. A major benefit of an architecturally neutral language is portability. The Java language explicitly defines any language feature whose implementation could be dependent on hardware or OS platforms. For example, Java specifies the size of primitives so that `int` in Java always refers to a signed two's complement 32 bit integer, whereas in C the size of `int` varies from platform to platform. Another example is the system libraries define portable interfaces. In the case of Java's abstract window class, for instance, there is a Windows implementation and a Unix implementation.

The benefits gained from Java's portability are numerous. Java bytecode can execute on any platform with a Java interpreter without modifying the bytecode from platform to platform. Furthermore, this portability coupled with the distributed properties of Java alleviate the burden of configuration management at the client. When modifications must be made to applets to correct bugs

or to provide new functionality, these changes only have to be made at the server. When a browser accesses a URL that points to a Java applet, the new version of the applet is downloaded from the server to the client's browser.

2.3.5.8 Interpreted. Java is an interpreted language which means that the bytecode produced by the Java compiler is converted into machine code and linked to other classes each time the application is executed. The bytecode stream contains additional compile time information that is not found in other languages such as C. This information is available to the interpreter at run-time and enables the interpreter to perform type checking and memory access validation such as ensuring array indexes are within bounds. These capabilities provide information concerning illegal memory access and a program trace facility for debugging. The Java interpreter also makes the linking process incremental; the bytecode is compiled and then it is linked to other classes at run-time. This incremental approach combined with the additional information provided by the compiler for the run-time environment reduces application development time by avoiding the code-compile-link cycle found with traditional compiled languages such as C and C++. With Java, many run-time errors are caught by the compiler or are prevented from happening because of Java's defensive nature.

2.3.5.9 High-Performance. Sun claims that the performance of the Java interpreter is usually more than adequate for most processing scenarios. The Java compiler performs register generation and does some optimization when it produces bytecodes from source files. Sun states that in tests conducted on a Sun SPARCStation 10 approximately 300,000 method calls per second were executed. Sun also states that the performance of compiled bytecode is almost indistinguishable from native C/C++ code.

2.3.5.10 Multi-Threaded. Most complex programs have to deal with several events, or threads, occurring simultaneously. Java is a multi-threaded language that can handle this type of situation. It provides a thread class which serves as the basis for all threads by including the methods for stopping, starting, interrogating (e.g., is it running or is it interrupted), interrupting and resuming threads. A multi-threaded application offers better interactive responsiveness and real-time behavior than a single-threaded application.

2.3.5.11 Dynamic Language. The Java language is designed to be dynamic so that it can adapt to an evolving environment and interoperate with other products. Take the following scenario that is commonly found in Unix environments. Vendor A produces a library of classes which support functionality that Vendor B's product uses, and both of these systems are installed separately on the same Unix system. If Vendor A makes changes to its library of classes, then Vendor B's product must be dynamic enough to take advantage of these new changes. Otherwise, Vendor B's product will not be able to take advantage of the new library, or the product may even fail. Java avoids this type of problem by linking classes at run-time when the interpreter loads and executes the bytecode. By linking classes at run-time, Java allows a developer to modify a class, e.g., adding methods and instance variables, without needing to change other classes that make use of the modified class.

Java also provides the capability to seamlessly integrate add-on utilities such as audio players, video players, and other file types to Web pages. The Web page can receive the Java applet, the media file and the means to view the file.

2.3.5.12 Java Disadvantages. As explained in the previous sections, Java has many advantages as a Web development language, but Java has some disadvantages as well. First of all, Java's specification is controlled by Sun and although it is widely supported, it is not an industry standard. As a result, there is a greater possibility that it may be replaced by another company's language than if it was accepted as a standard. Java is also a young language; it was initially released in November 1995. As a result, there is a limited set of third party Java development tools. Sun Microsystems provides an SDK which includes a compiler, debugger, class file disassembler, and interpreter, but these products are command line versions. It would be much more preferable, and necessary for large development efforts, to use GUI-based tools for development. Also, good documentation on and examples of Java are sparse, making development more difficult. These two problems will be less and less of a factor with time.

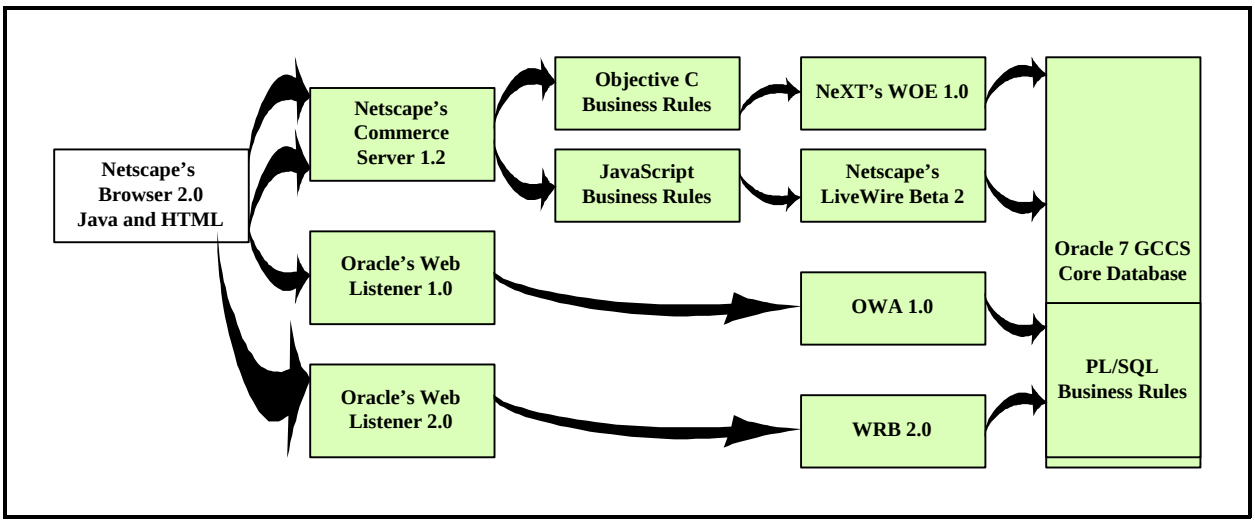
Another problem with Java is that while it offers more capabilities than HTML, it is also more complex to use than HTML. It is true that a large group of programmers, namely C and C++ programmers, already exists that should have a relatively easy Java learning curve, but there is another community that will find Java more difficult to learn: end users and casual programmers who know how to create HTML pages but have never learned formal programming features. It will take time for these developers to embrace Java, but the increased functionality of Java over HTML should speed up Java's acceptance and incorporation into Web systems.

There are also some security concerns with Java even though it is designed to provide a robust and secure environment. To date, many of the problems reported with Java's security are due to the Web browser. Recall from the security section that the browser has a responsibility to implement security in the form of what its Java interpreter will allow applets to do at run-time. In some cases, such as with the Netscape's Navigator 2.0 browser, bugs have been discovered. This problem relates back to the fact that Java is a very new language, and as a result, the products that make use of it are also very new. Another problem may occur when applets are written which spoof, or imitate, other applets. For example, suppose that a particular application includes a login window that requires users to supply a user ID and a password before they may see sensitive information. It would be possible to write an applet that imitates the login window to trick a user into providing the imitation applet a user ID and password. It must be understood that this type of threat may be minimized by other factors besides Java's own security. If, for instance, the environment in which the application exists is an Intranet, such as the SIPRNET, there may be other security measures in place such as secure servers where people would not have access to install a harmful applet in the first place.

In the prototype section, SRA points out some additional disadvantages of Java that were discovered by actually working with the language.

2.4 PROTOTYPE

An important aspect of SRA's evaluation was the the development of a prototype that used each of the Web application development tools being evaluated. The creation of this prototype is important because it afforded SRA's engineers and developers the opportunity to gain hands-on experiences with the different products, thereby allowing SRA to form conclusions that would not have been reliable or even possible based solely on a paper analysis. SRA selected S&M, a GCCS



Database application, to use in its prototype. Specifically, SRA implemented the Add Air Carrier thread of S&M. SRA prototyped this particular thread for a couple of reasons. SRA has used this thread in previous studies including the *JOPES Database Distribition (DBD) Alternatives Study* which contained performance data on the X protocol and SQL*Net using the Add Air Carrier thread and the *Distributed Computing Analysis Report* which implemented the Add Air Carrier thread using the Distributed Computing Environment (DCE). By using the same thread, it will be easier to make comparisons between different solutions, especially in the area of performance. Another reason for choosing the Add Air Carrier thread is that it incorporates a subset of the typical operations performed by a GCCS Database application. In particular, this thread requires the abiltiy to add several database records to a database as transactional modifications and to query data from a database.

SRA implemented the business rules and database I/O layers with three different products: Oracle's WebServer 1.0, NeXT's WOE, and Netscape's LiveWire. Additionally, in each of these cases, the business rules and database I/O layers were interfaced with the same Java GUI which provided the user interface and presentation layers. An Oracle7 database served as the DBMS layer. Figure 2-12 summarizes the different evaluation paths taken by SRA.

Figure 2-12: SRA's Evaluation Paths.

An important aspect of the prototype is the network environment in which the various evaluation paths were fielded. Figure 2-13 illustrates the network environment in which SRA fielded the prototype.

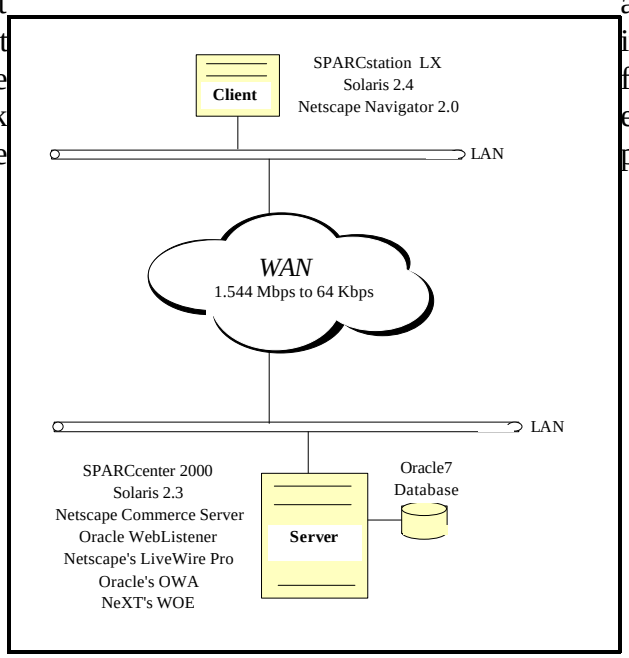


Figure 2-13: SRA's Prototype Network Configuration.

There are a few important points that need to be made about this configuration. First, notice that the client and server are connected via a WAN. This fact is important because it emulates the SIPRNET, the network on which GCCS applications are fielded. Furthermore, in the case of performance testing, this configuration allows the link speed between the client and the server to be varied from 64 kbps to 1.544 megabits per second (Mbps) to further emulate the SIPRNET. When dealing with applications that rely on network connections, obviously the link speed is an important parameter. Second, the use of one server indicates that the business rules, database I/O and DBMS layers reside on the same server. This fact, however, does not necessarily imply that these layers cannot be distributed to other servers.

The following sections discuss SRA's findings from the prototype. The discussions are organized by the evaluation paths shown in Figure 2-12 with the two Oracle paths grouped together. Java, however, has its own discussion and is only mentioned in the other three sections in the context of the Web application development products interfacing with it.

2.4.1 Evaluation Path 1: OWS 1.0 and 2.0

The business rules and data I/O layers of the prototype were implemented entirely in PL/SQL. The same PL/SQL stored procedures were used for both OWS 1.0 and 2.0, but the mechanism used to invoke them was different for each version. For the OWS 1.0 prototype, SRA used the OWA

CGI-compliant program described in Paragraph 2.3.2.2 to execute the PL/SQL stored procedures. For the OWS 2.0 prototype, SRA used WRBXs to execute the stored procedures. Initially, the HTP and HTF packages provided by Oracle were used to generate an HTML interface. Later, the Oracle PL/SQL procedures were interfaced with the prototype's Java interface. The OWS prototype involved approximately 1,600 lines of PL/SQL code. The entire OWS development effort took 20 staff-days, including time to learn the product.

Development with OWS proceeded rapidly mainly because the environment was easy to learn. It must be pointed out that the SRA developer of the PL/SQL stored procedures had previous experience with PL/SQL which no doubt decreased the learning curve. Procedure Builder, a component of the Oracle Developer 2000 tool set, was used to edit, debug, and compile PL/SQL. SRA feels that this product greatly enhanced the development environment and led to an increase in the speed of development by facilitating the code-compile-debug-correct cycle. For instance, to compile PL/SQL from within Procedure Builder, one need only make a single button click to save the changes to the database, and OWS automatically uses the updated code the next time the PL/SQL procedure is invoked. Procedure Builder also provides clear and helpful messages concerning compilation errors and automatically scrolls the display to the line in the code that caused the compilation error. Additionally, the OWA generates run-time error messages, stored in log files, that proved to be particularly useful.

The PL/SQL programming language provided an environment for implementing data I/O and business rules code rapidly because PL/SQL is relatively easy to learn. SRA's experience in the prototype indicates that programmers with a background in SQL will have an especially easy time picking up PL/SQL. For the prototype, a PL/SQL package was developed for each of the database tables accessed by the Add Air Carrier thread. These packages encapsulated the business rules and data I/O associated with each database table. The operations included syntactic and semantic validation of data, data insertions and predefined and dynamically created queries. Additionally, a package was developed for each of the GUI screens. These packages were responsible for passing data between the GUI and the business rules and data I/O packages.

Completing the Add Air Carrier thread demonstrated the excellent database access capabilities of PL/SQL which uses SQL and extensions to SQL. The majority of SQL code used in PL/SQL procedures, including the ones developed for the prototype, consists of compiled SQL rather than relying on dynamically interpreted SQL. This feature of PL/SQL often gives it a performance advantage over other programming languages that rely on dynamically interpreted SQL.

With all the advantages of PL/SQL, it must be noted that it is a specially designed language for implementing stored procedures and database triggers. As such, it has limitations not found in general-purpose programming languages such as C. Arrays with more than one dimension are not supported, and the only supported array indexes use integers. It is also only possible to perform data I/O with the database, and programmers are unable to call code written in other languages from PL/SQL. These limitations did not play a role in the prototyping effort as SRA was able to replicate the Add Air Carrier functionality of S&M. These limitations, however, may prove to be a hindrance in other development situations.

An important design decision that needed to be made to complete the prototype concerned the method used to return Geoloc Search results to the client. Queries performed using the Geoloc Search capability have the potential to return hundreds or even thousands of rows of data. There are three primary ways this data may be made available to the client's browser:

- Complete the entire query at the database server and then transmit all records produced by the query to the client at once.

- Complete a portion of the query, defined as a page, and transmit this page to the client. When the user requests a new page, redo the query, but only return the database records that constitute the next page. A page in this instance is merely a set number of records defined by the programmer.

- Perform the entire query but only return a page of records from the database. When the user requests a new page, return to the database query that has already been performed and retrieve the next row of data without redoing the query. Recall that when performing a query, a database application typically uses a cursor to store results. In order to avoid redoing the query, the application must have a way to maintain state information which associates a specific cursor with a specific user. In this way, when a specific user requests the next page of data, the application can access the cursor that was already being used to service that user's request rather than having to open a new cursor and redoing the query.

Notice that in the first method, the query is performed only once, and consequently, the overhead incurred from querying the database is minimized. This fact contrasts to the second method which must perform the query multiple times and thus, introduces additional overhead into the scenario. The second method, however, only sends data one page at a time instead of all at once as in the first method. Thus, the second method is less network intensive but more server CPU intensive than the first method. The third method is the most desirable because it supports the best capabilities of the first two methods without their drawbacks. In the third method, the query is performed only once, minimizing the use of server resources, and data is returned a page at a time, minimizing network traffic. In order to implement the third option, however, it is necessary to be able to keep track of maintain state information about the cursor at the server.

For the OWS portion of the prototype, SRA decided to implement the second method of sending data to the client for a couple of reasons. First, the OWS cannot keep state information for cursors, eliminating the third method from consideration. Second, OWS already included procedures that implemented functionality similar to the second method. In keeping with the goal of using a product's out of the box functionality to minimize the amount of custom code, SRA chose the second method. OWS implements the second method by returning state variables to the client. When a user performs a Geoloc Search, the first page of data is returned to the client by the OWS, along with the numbers of the first and last record returned. When the user makes a request for the next page of data, the client returns these numbers to the OWS. The OWS then performs the original query again and uses the last record number to return only the page of data that follows the last page. If a user wants a previous page, the OWS uses the first record number to return the previous page. This

technique is inefficient compared to solutions that avoid re-querying the database. The procedure that produces the next page of data is shown below.

```
procedure next_page ( first_record in integer,
                      last_record in integer,
                      where_clause in varchar2) is

      the_cursor integer;
      line varchar2(56);
      dummy integer;
      quote char(1) := '''';
      prefix char(8) := 'Data = '||quote;
      linefeed_char constant char(1) := CHR(10);
      suffix char(2):=quote||linefeed_char;
      sql_stmt varchar2(600)
      row_count integer :=0;
      wrote_one boolean := false;

begin

      -- build a SQL statement that returns geolocation data
      -- in format for use by Java applet.  Use the input
      -- where_clause
      sql_stmt :=  'select  glc_nm||''|''||
            cy_st_nm||''|''||int_cd||''|''||glc_cd||''|''||
            ap_icao_fac_cd||''|''||milstamp_id  from
            geohelp_view'|| where_clause;

      //The query is redone each time.
      the_cursor := dbms_sql.open_cursor;
      dbms_sql.parse (the_cursor,sql_stmt, dbms_sql.native);
      dbms_sql.define_column (the_cursor, 1, line, 56);
      dummy := dbms_sql.execute (the_cursor);

      //Only the rows that constitute the next page are sent back to the
      //client.
      loop
            if dbms_sql.fetch_rows (the_cursor) > 0 then
                  row_count := row_count + 1;
                  if (row_count >= first_record) then
                        if (row_count <= last_record) then
                           -- the record is in the range
                           if not wrote_one then
                            -- write success msg before 1st record
                             htp.p('Message = ''Success''');
                             wrote_one := true;
                           end if;
                           dbms_sql.column_value
                              (the_cursor, 1, line);
                           -- write the record back to the applet,
                           -- adding 'Data =' and a linefeed
                           htp.p(prefix||line||suffix);
                        else
                           exit;
                        end if;
                  end if;
            else
                  -- no records were fetched
                  exit;
            end if;
```

```
        end loop;

        if not wrote_one then
              -- return an error message to the applet
              if first_record = 1 then      -- initial query
                    htp.p('Message = ''No geolocations matched the search
                    criteria''');
              else
                    htp.p('Message = ''No more records''');
              end if;
        end if;

        -- return the where_clause back to the applet.
        htp.p('Clause = '||quote||where_clause||quote);

        dbms_sql.close_cursor (the_cursor);


end next_page;
```

SRA prototyped the Web Listener's basic authentication. Upon accessing a specified URL for the first time in a session, the user is presented with a username/password dialog box. When the user types in a username/password combination, the Web Listener validates it against a list of valid username/password combinations stored in its configuration files. If valid, the Web Listener spawns a WebAgent CGI or WRBX process to handle the request. SRA, attempted to use digest authentication, but it would not work with the Netscape's Navigator 2.0 version browser which does not support it. SRA did not prototype any of the SSL 2.0 features offered by the OWS 2.0 Web Listener.

2.4.2 Evaluation Path 2: NeXT's WOE

The NeXT WOE prototype was implemented using a combination of WebScript and Objective C. The user interface was first implemented using WebScript to generate HTML pages. SRA then converted the prototype to receive its user input from the Java GUI. The entire effort took 50 staff-days. SRA wrote about 2,100 lines of WebScript and Objective-C code for the WOE prototype. WOE proved to be a complex and ambitious product to master, but once learned, it provided a robust environment for developing code in a reusable, maintainable manner.

The prototype effort indicated that WebScript is best suited for the development of the portion of the application responsible for receiving input from the GUI and for preparing output to be sent to the GUI for display. Furthermore, SRA noticed WebScript's lack of string manipulation capabilities, preventing its use in the development of some business rules. In these cases, SRA used the Objective C programming language to code the business rules. For example, Objective C code was developed for the Geolocation Search capability to perform a simple character substitution operation which could not be done in WebScript. The lack of clear runtime error messages hindered development of WebScript. Often, the only message produced by the WebScript interpreter after an error is "erroneous statement" which was not very helpful with locating and correcting errors. Between the two languages, however, there was more than enough capabilities to replicate the functionality of the Add Air Carrier thread.

As previously stated, Objective C is a fully object-oriented extension to the C language. The prototype effort revealed that it is much closer to standard C than is C++. Therefore, a C programmer should have a much easier time mastering Objective C than C++. The compiler provided by NeXT is the GNU C compiler. It is capable of compiling C, C++, and Objective C and comes with a debugger. Linking a WOE application with the EOF and with other Objective C code proved difficult at first because the procedure for linking is not well documented, but this problem was eventually overcome with help from NeXT's engineers.

The EOF was used by the WOE prototype to interface with Oracle. In particular, the EOF interacted with Oracle by dynamically generating SQL statements. These dynamically generated SQL statements are generally slower than the compiled SQL statements used by the OWS. The EOF allows interaction with a database using any combination of three distinct interfaces, each providing a different level of abstraction from the DBMS: the data source layer, the database layer, and the adaptor layer. The data source layer is the simplest and most abstract of the three levels. At this layer, logging into the database and performing a query requires only four statements, but all the records from the query must be retrieved, or fetched, from the database at once. The database layer allows the use of cursors when fetching so that a single record may be fetched at a time until no more records are found. Performing a query in this manner requires about 10 statements. The lowest level, the Adaptor layer, has two primary functions. First, it converts code written at the data source and database layers into code that makes use of a DBMS's native CLI. Second, it allows programmers to write DBMS-specific SQL statements. It is more desirable to use the adaptor layer in the first capacity because it leads to a reduction in the amount of code that must be written in a heterogeneous DBMS environment. As a result, SRA used the adaptor layer to convert the more abstract statements at the data source and database layers to native CLI-specific code.

In order to make use of WOE's ability to fuse database data with methods, SRA had to model the database used in the prototype. SRA used NeXT's NEXTSTEP-based EO Modeler to create a model, an ASCII file describing the database schema, of the seven GCCS Core Database tables and views used in the prototype. The EO Modeler also generated an Objective C class for each table. Each class contained a declaration for every data item in the table, and methods for setting and getting each data item. It served as a first step, or template, in the development of object methods that are business rules.

As in the OWS prototype, an important aspect was the implementation of the Geoloc Search functionality. The WOF has the capability to store session state regarding cursors at the server, allowing SRA to implement the third option for returning data to the browser. Recall, in this option, the query only has to be performed once, but data can also be returned a page at a time. This option conserves server and network resources. The WOF uses session variables to maintain state which it creates when a session is initiated and makes available to an application when subsequent requests are made. A session is initiated when a user submits a URL that points to a WebObjects application. Once a WOF application is started, it never terminates, unless instructed to terminate by the programmer, so the values of session variables are never lost. A WOF application can maintain session variables from multiple simultaneous sessions by using a unique session ID for each client that is returned to the client when a request is made. When the client makes subsequent requests, it also returns the session ID to the WOF, allowing WOF to access the proper session's variables.

This capability is applied to the implementation of the Geoloc Search in the following manner. The query results are stored in a session variable known as a data source. When the initial query is made, the application stores the results in a session variable and returns only the first page to the client. When the client makes a request for the next page, WOF accesses the original data source to get the next page of records. The following WebScript code is executed to initialize a data source that logs into the database at the beginning of a session.

```
session id geolocSource; // make Geoloc data source a session variable

/*    Name: didPrepareForRequest
      Called by WOF when Geolocation Search URL is requested
      Purpose: Determines if  this is the first request in the
            session.  If so, it initializes a Geolocation
            data source
*/
- didPrepareForRequest: theRequest inContext: theContext
{
// check if a session variable has been initialized
if  (!geolocSource) {

      /* if not, this is the first request from this user
         initialize a Geolocation data source */
      geolocSource = [[EODatabaseDataSource alloc]
            initWithModelName:@"GCCS"
            entityName:@"geohelp_view"];
      if (!geolocSource)
            [self logWithFormat:@"Error: database not
                  accessible"];
}
```

In the preceding code, `geolocSource` is the data source, and it is used by a method called doQuery that actually performs the query and stores the result. A different method, nextPage, then accesses `geolocSource` during subsequent requests to return the next page of records. The main point to understand is that each time the nextPage method accesses `geolocSource`, the doQuery method does not have to redo the query. The nextPage method is part the component script shown below.

```
/*    Name : nextPage
      Called by: WOF
      Purpose:    Returns the next page of geolocation records in the
            array "rows".   "rows" is mapped to a dynamic WebObject
            which is returned to the Java applet by WOF */

id rows;


- nextPage
{
id count=0;
id aGeoloc=nil;
id status;
id row;

// initialize the array "rows"
rows=[[[NSMutableArray alloc] init] autorelease];
```

```
// check geolocSource to see if  a query is in progress from a
// previous call to doQuery.  Because geolocSource is a session
// variable it is accessed from the application script using the
// syntax [WOApp geolocSource]
if (![[[WOApp geolocSource] databaseChannel] isFetchInProgress]) {

      // return error message to applet
      [rows insertObject:@"Message = 'No query in progress'"
            atIndex:0];
      return self;
}

// get the next “lines_per_page” records from geolocSource
while ((aGeoloc = [[[WOApp geolocSource] databaseChannel] fetchWithZone:
nil]) && (count < [WOApp lines_per_page]) ) {

      // get each data item in the record and
      // put the data in the format readable by the applet
      row = @"Data =";
      row = [row stringByAppendingFormat:@" '%@",[aGeoloc
objectForKey:@"Glc_Nm"]];
      row = [row stringByAppendingFormat:@"|%@",[aGeoloc
            objectForKey:@"Cy_St_Nm"]];
      row = [row stringByAppendingFormat:@"|%@",[aGeoloc
objectForKey:@"Int_Cd"]];
      row = [row stringByAppendingFormat:@"|%@",[aGeoloc
objectForKey:@"Glc_Cd"]];
      row = [row stringByAppendingFormat:@"|%@",[aGeoloc
objectForKey:@"Ap_Icao_Fac_Cd"]];
      row = [row stringByAppendingFormat:@"|%@'\n",[aGeoloc
      objectForKey:@"Milstamp_Id"]];

      // put each record  in the array “rows”
      [rows addObject:row];
      count++;
}

// check if any records where found, return appropriate message
// to applet
if (count == 0)
      [rows insertObject:@"Message = 'No data found'" atIndex:0];
else
      [rows insertObject:@"Message = 'Search Successful'"
            atIndex:0];
}
```

Another important point is WOF applications are single-threaded, meaning they handle requests one after another, in a first-in-first-out fashion. If better performance is desired in a multi-user environment, WOF can be configured to run more than one instance of a WOF application at once. Incoming requests can then be serviced by multiple applications.

WOF also supports maintaining session state on the client. All session variables can be encoded into a hidden HTML field which is sent to the client after each request is serviced. The client then returns this data to WOF when making subsequent requests. SRA did not prototype this capability.

2.4.3 Evaluation Path 3: Netscape’s LiveWire Pro

For the Netscape LiveWire evaluation path, SRA implemented the business rules and data I/O layers of the prototype in the form of source files that were HTML files with embedded JavaScript. The JavaScript compiler was then used to compile these source files and to produce executable .web files. A .web file, or LiveWire application, was developed for each set of business rules and data I/O, e.g., adding an air carrier and performing a Geoloc search and OPLAN search. The LiveWire development environment was easy to learn, and SRA was able to write all the business rules and data I/O code in 10 staff-days. For the prototype, SRA wrote approximately 450 lines of JavaScript code. Notice that the LiveWire prototype involved the least amount of code and time to finish. There are a couple of factors that contribute to this fact. First, the LiveWire prototype does not include the same level of functionality as the OWS and WOE prototypes. Notably, SRA did not implement any paging for the Geoloc Search functionality as it did for the OWS and WOE prototypes. SRA's implementation for returning geolocation codes will be discussed later in this section. Additionally, the error handling used in the LiveWire prototype is not as extensive as the error handling used in the other two prototypes because SRA was unable to ascertain how to make use of Oracle server messages with LiveWire. LiveWire should be able to use Oracle server messages, but LiveWire's documentation on this area was insufficient for SRA to make use of them. Increasing the functionality of the LiveWire prototype in these two areas would certainly involve additional time and code.

A second reason for the smaller amount of code and development time for the LiveWire prototype is its use of JavaScript. Recall from the Netscape analysis section that JavaScript is designed to simplify the complexity of programming, allowing HTML developers without experience with traditional programming languages such as C to increase the functionality provided by their HTML pages. JavaScript achieves this simplification by including as easy to learn syntax, specialized built-in functionality, and minimal requirements for object creation. PL/SQL, Objective C, and Java are designed to be full-featured programming languages. Java for example allows developers to extend classes and supports the creation of multi-threaded applications. With increased functionality, however, comes increased complexity, and this complexity leads to additional code and development time. Furthermore, it must be pointed out that the SRA developer of the JavaScript code had previous experience with C which is syntactically similar to JavaScript.

Working with the JavaScript language revealed that JavaScript is a loosely typed language in comparison to Sun's Java. As a result, the JavaScript compiler does minimal object type checking during compilation, and the LiveWire run-time environment verifies the correctness of an access at run-time. The major consequence of this behavior is that many simple errors can be passed through the implementation cycle, avoiding detection until the testing cycle. A simple example follows to illustrate this point. Consider the simple JavaScript function shown below:

```
<server>
function foo()
{
for (I=0; I <10; I++)
      {
      writeln("I am in foo and the index is ", I);
      }
}
</server>
```

This function would compile without errors and produce 10 lines of output. Now, consider what would happen if the function contained a simple misspelling as in the following example:

```
<server>
function foo()
{
for (I=0; I <10; I++)
      {
      writenl("I am in foo and the index is ", I);
      }
}
</server>
```

In this example, writeln has been misspelled as writenl. When this function is compiled, the JavaScript compiler fails to notice the misspelling and generates the .web file. The error will not actually be detected until the function is run, complicating the process of fixing errors.

As in the other prototype, an important decision was how to return data generated by the Geoloc Search functionality. In the OWS prototype section, three methods for doing this were introduced:

- Do the query once and return all the data to the client.
- Do the query once and return a page of data to the user. When the user requests the next page of data, redo the query and return the next page of data.
- Do the query once and return a page of data. When the user requests the next page of data, use an already existing cursor to return the next page of data without having to redo the query.

The third method was not used because LiveWire does not support state management for cursors. The second method was not chosen because unlike the OWS, LiveWire does not offer any out of the box implementation for this type of paging. Consequently, SRA used the first method as implemented by the following code.

```
<server>
if(!database.connected())
{
database.connect("ORACLE", "BET4.WORLD", "TABLEMASTER", "MASTER", "");
}

if (!database.connected())
{
write("Message = 'Error: Unable to connect to database.'\n");
}
else
{
query = "select
geographic_location.glc_nm,country_state.cy_st_abbrd_nm,geographic_locat
ion.int_cd,geographic_location.glc_cd,geographic_location.ap_icao_fac_cd
,geographic_location.milstamp_id from geographic_location, country_state
";
```

```
var NumRows = 0;

while (q.next()) {
      NumRows = NumRows + 1;
      if (queryinit == true) {
            write ("Message = 'Success'\n");
      queryinit = false;
}

write("Data = '" +  q[0] + "|" +  q[1] + "|" +  q[2] + "|" +
    q[3] + "|" + q[4] + "|" +  q[5] + "|" +  "'\n");

}
if (queryinit == true) {
      write ("Message = 'No Records Retrieved'\n");
}
q.close();

database.disconnect();
}
</server>
```

SRA encountered some problems during the development of the LiveWire prototype. A major problem occurred when SRA registered too many applications with LiveWire. In order for the Commerce Server or Enterprise Server to execute LiveWire applications, the applications must first be registered with the LiveWire Applications Manager. Additionally, in the prototype, SRA used a CGI script to pass parameters to the Java GUI. SRA found that if too many applications were registered, i.e., more than 11 or 12 applications, the Commerce Server was unable to execute the CGI script because it was unable to open an interprocess communication pipe between itself and the CGI process. Recall that for CGI, the HTTP server must stay in contact with the CGI process. Netscape did not have an explanation for why this problem was occurring. SRA investigated several different possibilities for remedying it, but the only action that solved the problem was decreasing the number of registered LiveWire applications. Obviously, this is an unacceptable solution, and Netscape needs to fix this problem in its production release of LiveWire.

SRA also experienced problems with actually registering LiveWire applications using the LiveWire Applications Manager. Specifically, the Application Manager allows applications to be stopped and restarted so developers can make changes to them. This feature did not work and resulted in a core dump whenever SRA attempted to use it. SRA had to bypass this problem by shutting down and restarting the Commerce Server whenever a LiveWire application was modified. This work around was adequate for the prototype environment, but it would absolutely not be satisfactory in a real world environment. Additionally, each LiveWire application had to reside in its own directory, or the application did not function properly. These problems indicate that Netscape has some fairly major errors to correct between now and the release of its production product.

2.4.4 Java

Java was used to implement the prototype's GUI for several reasons. First, the creation of this prototype offered an excellent opportunity to assess the capabilities of Java. Many claims have been made about the Java language and programming environment that make it suitable for the development of large scale Web-based applications. Second, Java offers some inherent capabilities

that go beyond those found in HTML. To better illustrate this fact, take the example of a Web search engine that uses a CGI compliant program to perform searches. Figure 2-14 shows what the search criteria screen may look like.

Figure 2-14: HTML Search Screen.

Notice that the user has specified Java in the search window. When the user presses the search button, the query is sent back to the server, and the CGI compliant application is executed to perform a search Java related URLs. After performing the search, the CGI application formats the results into another HTML page. This page is then served to the client's browser which displays the

results. This example illustrates an important fact; each time the user needs to see a new HTML document, whether it be statically or dynamically created, a new page must be served by the server for display by the client browser. This process is inefficient and difficult to use from a user's point of view. In the case of a more complex application such as the Add Air Carrier thread, this problem becomes more pronounced. The Add Air Carrier thread also has subthreads such as Geoloc Search. With HTML, the user inputs for providing the Add Air Carrier information would be provided by a different HTML document, or page, than the one that provides the user inputs for specifying Geoloc queries. Users would have to navigate back and forth between the documents. From a user's and a programmer's point of view, it would be difficult to populate the Add Air Carrier inputs with geolocation codes from the Geoloc Search page. Java, however, provides the ability to open windows that run within the applet. As a result, the Geoloc Search user interface may be implemented as a pop-up window that is a part of the Add Air Carrier interface rather than as a separate HTML page. In this way, a new page does not have to be served when the user wants to perform a Geoloc Search, and the Geoloc Search window may also pass data to the Add Air Carrier window. Java also supports more functionality than HTML in terms of user interface widgets such as buttons, scrolling list boxes, text entry fields and choice menus.

SRA used the Java development environment, or System Development Kit (SDK) version 1.0, supplied by Sun Microsystems in the prototype. The SDK contains:

- The Java compiler, javac, which translates Java source code to platform independent bytecodes
- The Java interpreter which executes Java programs on the end-user's PC or workstation
- The C header file, javah, and source file, java_g, generators which produce header files for C as well as source files for creating methods within Java
- The Java disassembler, jalap, which prints out information about class files, a file containing a Java applet
- A document generator, javadoc, which can produce HTML files from Java source code files
- A java applet viewer, allowing programmers to preview their applets without having to use a browser
- A profiling tool, javaprof, which is used to format statistics on run-time performance data that can be obtained for Java classes.

Additionally, SRA's developer used several standard Unix tools such as the make utility. Sun's SDK does not include any visual development tools or an Integrated Development Environment (IDE). Initially, this lack of advanced development tools would appear to hamper the development process, but as will be discussed, this absence did not greatly hinder the prototype development process.

The process used to develop a Java applet, in this case the Add Air Carrier thread GUI, followed the traditional software development process of code, compile and test. The Java compiler proved to be defensive in that it detected questionable code that would have generated run time errors. This same code would have been passed by a regular C/C++ compiler. Examples of frequently caught errors were uninitialized variables and invalid type casts.

Debugging the applet was simple and painless considering the tool set used for the job was relatively primitive. In fact, the only method used to debug the applet was inserting print statements into the applet code at points where problems were suspected. As previously mentioned, Java is an interpreted language which reduces the amount of time required for compilation, allowing the compile, test, correct cycle to be repeated many times. An additional feature of the language that proved useful in the prototype development was its stack dumps. Stack dumps were useful because they identify the source file and the line number in the source file where an error has occurred. In most cases, the stack dump information was sufficient to isolate the incorrect code and to make the correction usually with a single iteration of the correct, compile and test cycle. In short, although Sun's SDK includes a debugger, SRA did not encounter any errors that required the power of a debugger.

A significant problem that was encountered during development was the Netscape browser's inability to load applets that had been changed, or updated, at the server. Instead, Netscape's browser loaded applets from its local cache and thus, did not always load the latest version of the applet. SRA solved this problem by reducing the browser memory and disk caches to zero. This action helped speed up testing of the GUI applet.

The most time consuming part of the Java prototype was the design and implementation of the presentation layer, or the code responsible for actually painting the windows and widgets. This code is repetitious and is a sizeable portion of the prototype code, approximately 30% of the entire GUI code which also includes the user I/O code. The presentation portion was the most time consuming aspect of the GUI development process. A significant amount of time was used to make changes to the code to determine if the prototype windows could be made to have a certain look and feel. This experimentation was done by trial and error. The prototype consisted of three screens, but for a large application development project with many screens, this trial and error method would be a major drawback. Several what-you-see-is-what-you-get (WYSIWYG) GUI designer products have been released or are in beta testing. If a major development effort is undertaken to implement GCCS applications using Web technology, it would be a worthwhile effort to identify an appropriate GUI designer tool. The entire GUI code, presentation, and user I/O layers, consisted of:

- € 25 source files containing
- € 38 classes
- € 2,234 lines of code and comment
- € 1,325 lines out of the 2,234 are for the GUI
- € 64.6 kilobytes of bytecode.

A decision by Sun's Java development team was to base Java on the C and C++ language and thereby provide a large base of experienced developers to allow companies to begin Java development without a major investment. SRA's experience from the prototype indicates that C

developers with object-oriented programming experience will have a minimal Java learning curve. The difference between the Java language and C/C++ is minimal; both languages have identical syntax, basic data types, literals, operators and statement construction. The major difference is Java's lack of the American National Standards Institute's (ANSI's) standard library containing functions that C/C++ developers are accustomed to using. This lack of the ANSI standard library is not a factor because the Java class library provides much of the same functionality that the ANSI library provides. C developers with no object-oriented programming experience would have a greater learning curve, but SRA is unable to quantify this increase because SRA's prototype developers have object-oriented experience.

The Java development environment was not without problems. It became quickly apparent there was a lack of documentation. The Java class library is documented with an online reference guide, but this guide is terse and lacked sufficient examples and explanations to make full use of the Java class library. Figure 2-15 provides an example of this documentation.

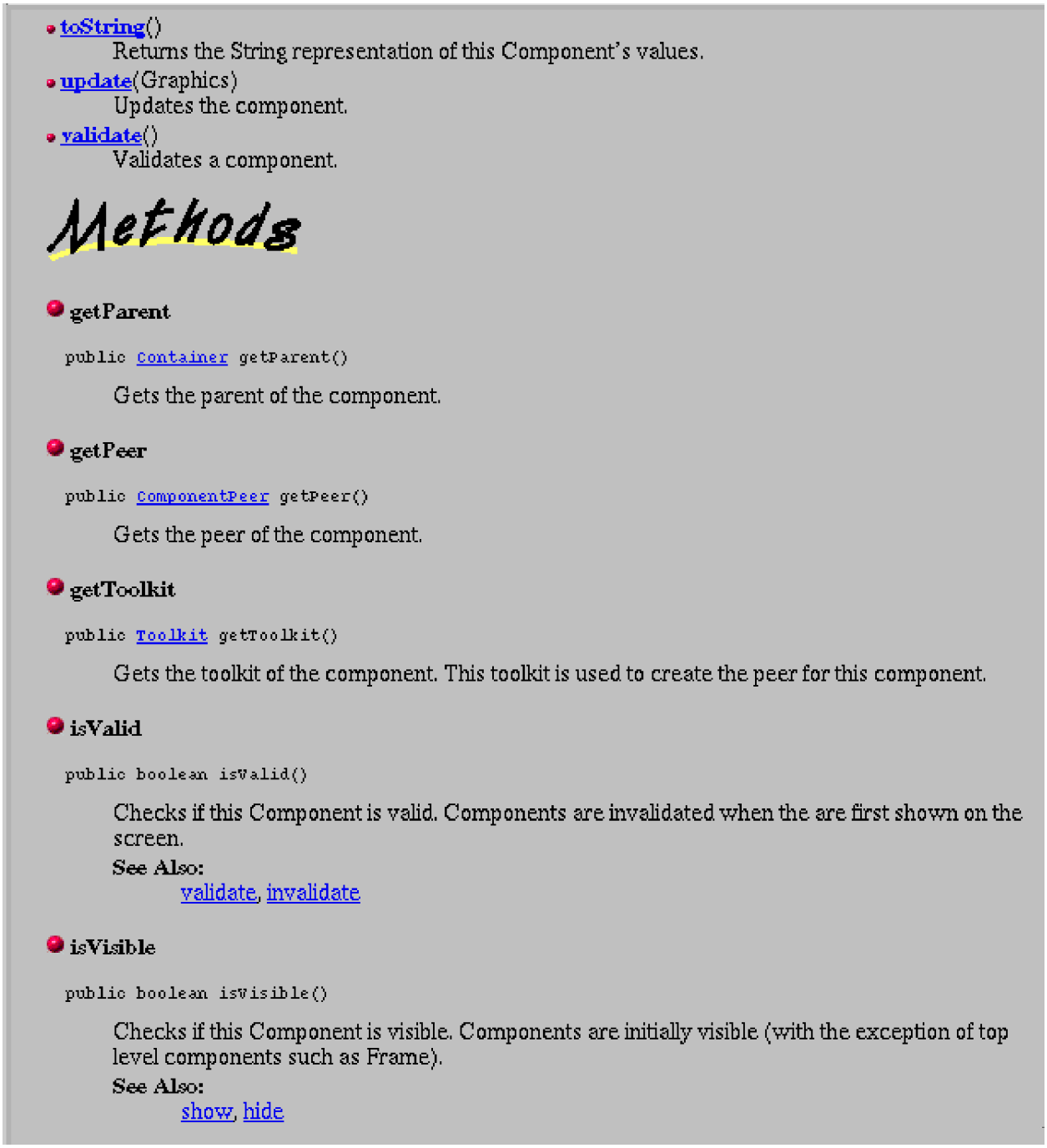


Figure 2-15: Document Example.

The Java class libraries contain 111 classes which among them contain 286 class variables and 1,729 method calls, most of which are documented by a single line of text as shown above. To use the Java libraries to their fullest extent and thus reduce the amount of code, more detailed

documentation is necessary. SRA estimates that as much as 15% of the prototype code could be eliminated by making better use of the Java class library.

Another major problem uncovered by the prototype concerned Java's Abstract Window Toolkit (AWT). The AWT is responsible for handling display windows and their associated objects. The number one problem encountered with the AWT was the inability to place objects at specific locations on the window. A second problem with the AWT was that some display objects did not work properly. Some examples of these problems that were encountered during the prototyping effort are:

- On the Windows 95 platform, buttons were drawn in the default color, gray, even when a different color was explicitly specified
- Placing greater than 20 items into a choice menu froze the machine
- Placing choice menus at certain locations would froze the machine
- Choice menus, regardless of platform, were drawn in the default color, gray, even when a different color was explicitly specified
- On the Windows 95 platform, the scroll bars for a list box were not displayed
- On the Windows 95 platform, text was drawn in the default color, gray, even when a different color was explicitly specified.

Some of these errors are minor presentation problems, while others are more severe, actually preventing the Java applet from functioning. These errors also bring up another topic: Java's dependence on the Web browser. In order to make use of Java applets, the Web browser must have a Java interpreter. The Java interpreter is platform dependent. For example, the Java interpreter for a Netscape Windows 95 browser did not draw the prototype screens the same as the Netscape Solaris browser. Instead, the Java interpreter behaves a little differently so that SRA observed that a Netscape Windows 95 browser always displays buttons as gray, whereas the Netscape Solaris browser displayed buttons as the color specified by the programmer. This problem stems from Java's youth as mentioned in the Java analysis section and the interpreter's dependence upon the native windowing system. As time passes and vendors begin to incorporate Java support more extensively into their products, many of these inconsistencies should be eliminated.

Although Java and C/C++ do share a common syntax and language structure, Java contains some additional features that were encountered during the prototype and deserve mention. The first such notable feature is try/catch statements for exception handling. In C/C++ the typical code loop for handling an error would look like the following:

```
int  handle = open("data.file", O_RDONLY);      //  create  a  read  only
                                                //  file pointer
char buffer[1000];                              //  buffer  for  data  in
                                                //  the file
if (handle == -1)
      {
```

```
        printf("Could not open file.  Try again later.");
        }
else if (read(handle, buffer, sizeof(buffer)) == -1)
        {
        printf("Failed to read data file.  Try again later.");
        close(handle);
        }
else
        {
        // do whatever processing is required
        }
```

Using the try/catch statements provided by Java, the above statements would look like the following:

```
int  handle;            // create a read only file pointer
char buffer[1000];      // buffer for data in the file

try
        {
        handle = open("data.file", O_RDONLY);
        read(handle, buffer, sizeof(buffer));
        // do whatever processing here
        }
catch (IOException IoError)
        {
        printf("An i/o error occurred");
        }
```

In the above example, the statements in the `try` block are executed, and if an error occurs, the statements in the catch block are automatically executed. A brief comparison between the standard C/C++ blocks and the Java block shows that the Java structure is cleaner because the method's logic steps are separated from the error handling which in turn makes the code easier to understand and simpler to maintain.

The Java development environment provides a rich class library containing support for I/O, network communications in the form of FTP and HTTP and multiple thread processing. Using threads in an applet is straightforward, provided the applet is structured to take advantage of multiple threads. SRA, however, encountered a bug when attempting to make use of multiple threads. The Java class responsible for interrogating threads, e.g., see if a thread has completed its job, was not functioning properly. Sun claims that this bug is due to a problem with the Netscape interpreter. SRA's prototype supports this claim because Sun's applet viewer allowed multi-threading to occur properly.

A major capability missing from the Java class library is the capability to perform contiguous I/O operations over a single connection. All of the file I/O and network I/O classes expect input as a continuous stream. Once the end of the stream is detected, the applet closes the connection. So, for example, if an applet sends information to another application and then needs to wait for a return message, the applet must be forced to wait and to continuously check the connection to see if new data is being transmitted. SRA worked around this problem by creating the following essentially infinite loop to continuously retry the connection:

```
while ((nowDate.getTime() - startDate.getTime()) < 60000)
```

```
            {
            if ((inputBuffer = inStream.readLine()) != null)
                {
                results.addElement(inputBuffer);
                System.out.println(inputBuffer);
                if (inputBuffer.startsWith("Message") == true)
                   {
                   break;
                   }
                }
            nowDate = new Date();
            }
        }
    catch (Exception ioException)
        {
        results.removeAllElements();
        results.addElement("Message = '" + ioException.getMessage() + "'");
        }
```

An issue under consideration is the security capabilities provided by the Java language. This cannot be separated from the capabilities provided by the Netscape browser because the applet is executing within the browsers control. As previously discussed, Java provides three levels of security protection, ranging from no protection to a strict protection level, where an applet can only access the server from which it was loaded. The Netscape browser implements the strictest level of security protection, allowing applets access to only the server from which it was loaded.

2.5 PRODUCT COMPARISONS

This section compares the products against each other in terms of their ability to satisfy the requirements described in Paragraph 2.2. For Oracle, the evaluations are for OWS 2.0 as this is Oracle's leading product. These comparisons are based on the findings presented in the analysis and prototype sections. For each requirement area, functional and non-functional, an evaluation is provided which indicates how well each product satisfies the particular requirement. An evaluation of Java is also included when appropriate. A €+ indicates the product exceeds the requirement or provides enhancements. A € indicates the product satisfies the requirement. A €- indicates the product does not fully satisfy the requirement, or when possible, significant development is required to satisfy the requirement. If the product does not satisfy the requirement in any way, an € is shown. A € indicates that SRA prototyped the requirement.

2.5.1 Access to GCCS/JOPES Data

For this requirement area, an evaluation for each specific requirement is included to show differences in the products that may not be evident if the area is evaluated as a whole.

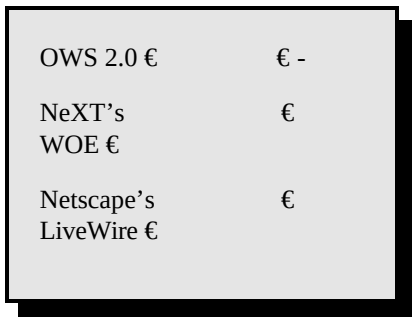
2.5.1.1 Accessing Oracle, Sybase and Informix Databases.

All three products are able to access data residing in Oracle, Sybase, and Informix DBMSs. SRA actually prototyped each product accessing an Oracle7 database.

2.5.1.2 Performing Transactional Modifications, Queries and Rollbacks.

Each of the products demonstrates the capability to access GCCS/JOPES data in the two ways described in Paragraph 2.2.1.1: transactional modifications and queries. Furthermore, each product can interface with the same Java GUI, allowing a user to specify data for transactional modifications and for retrievals. Additionally, each product supports the nullification of an entire transactional modification if any part of the modification fails.

2.5.1.3 Database Access Method.



Netscape's LiveWire and NeXT's WOE receives the highest mark for this requirement because of its transparent support of published, native CLIs and ODBC. When accessing Oracle, Sybase, or Informix databases, they use the applicable native CLI, i.e., OCI, DB-Lib/CT-Lib or ESQL. When accessing a MS SQL/Server on NT databases, they use ODBC calls as the access method. In any case, the method used is transparent to the developer.

Oracle receives the lowest mark for one primary reason; they do not support the published, native CLI for any of the databases, including its own. They use ODBC for accessing Sybase and Informix however, the developer does not have the option of taking advantage of the native CLI which provides more functionality and in some cases better performance than ODBC. In the case of its own database, Oracle uses its unpublished CLI to access the database.

2.5.1.4 Changing Databases.

OWS 2.0	€
NeXT's WOE	€
Netscape's LiveWire	€

All the products support functionality that allows a user to access different databases programmatically or to use a default database. The way this is done is an implementation decision. None of the products provide functionality that stands out from the other products' functionality for this requirement.

2.5.2 Process Distribution

OWS 2.0 €	€ -
NeXT's WOE €	€
Netscape's LiveWire €	€
Java €	€

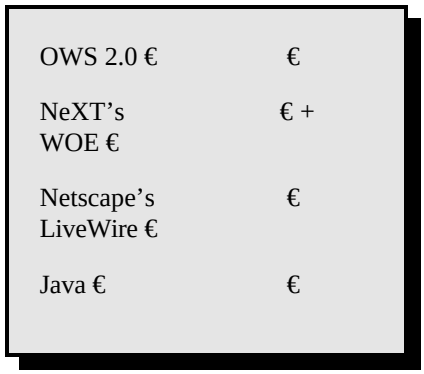
For this evaluation, SRA prototyped one particular configuration, a client connected to the server via a WAN, but other configurations are possible. Each of the web application development tools and Java allow system software layers to be implemented in a modular fashion, and all developed code is supported by the COE hardware and OS platforms. Oracle, however, receives a lower rating for this requirement because of its reliance on PL/SQL stored procedures. In order to separate the data I/O layer from the DBMS layer, Oracle requires an additional database instance to store the PL/SQL. Needing this instance increases the burden of configuration management of infrastructure products and uses server resources, eliminating some of the benefits of a modular system.

2.5.3 Product Platform Independence

OWS 2.0 €	€
NeXT's WOE €	€
Netscape's LiveWire €	€

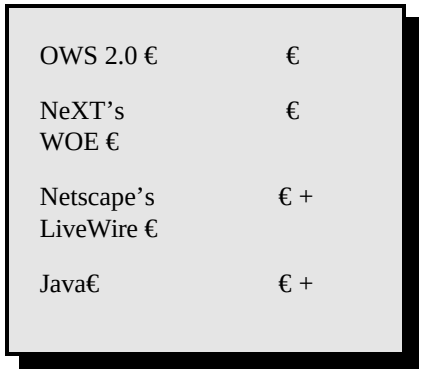
Each of the web application development tools support the COE hardware and OS platforms as well as some additional platforms. SRA fielded the prototype in a Solaris environment running on a SPARC hardware platform. The use of the HP-UX and Windows NT OSs was not prototyped.

2.5.4 Product Interoperability



All the development products satisfy the minimal requirement of interoperating with the Netscape Navigator Web browser. WOE, however, is given the highest mark for two reasons. First, it interoperates with any HTTP server that supports CGI, and second, using PDO, it provides interoperability between its own enterprise objects and OLE/COM objects.

2.5.5 Configuration Management

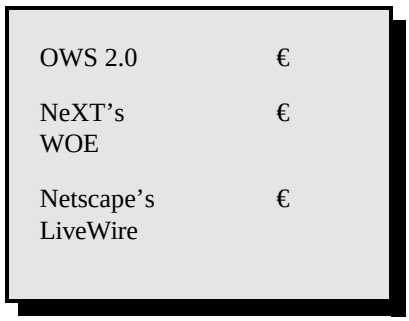


SRA evaluated configuration management in two areas: ease of installing products and ability to maintain and update code. In terms of installing the products, SRA was able to install each product with minimal difficulty. One of the primary advantages of using the web environment is it eases the burden of maintaining and distributing new versions of software. This advantage is reflected by the fact that two of the four products received the highest mark possible. The OWS 2.0, LiveWire and Java allow programs to be written that are in effect platform independent, i.e., they do not have to be recompiled for different platforms. OWS achieves this independence by using PL/SQL and Java. Once PL/SQL is compiled, it resides in the Oracle database and can be distributed to any platform on which Oracle's database can run without recompiling the code. The OWS was not given a check plus because of the added configuration work required to support the instance of the Oracle database that is required to store the PL/SQL. Java and LiveWire, through JavaScript, achieve independence by compiling source code into bytecode. Additionally, Java alleviates the burden of configuration management at the client because applets are stored on a small number of servers as compared to storing them on a larger number of clients. WOE received a lower mark because it depends on Objective C for the implementation of some business rules. Objective C must be recompiled for each platform on which it will run. SRA developed code for the Solaris OS running on the SPARC hardware platform and did not have the opportunity to test the portability of the business rules and data I/O code for the HP-UX and Windows NT OSs. In the case of the GUI, however, SRA was able to download and run identical Java applets on Solaris 2.4 and Windows 95 clients.

2.5.6 Duplication of GCCS Database Applications' Functionality

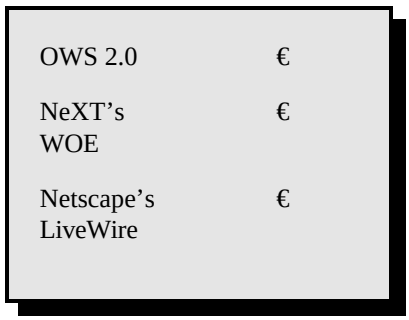
The duplication of GCCS Database applications' functionality is an important requirement area. Each of the products is able to fulfill this requirement for the particular thread prototyped by SRA, the Add Air Carrier thread of S&M.

2.5.7 Security



This requirement is evaluated only to the extent that the different products support the Unix and database authorization and authentication required for users to access GCCS applications and to make changes to the GCCS/JOPES Database. Each of the products can satisfy this area. This analysis does not address the Defense Information System Agency's (DISA's) advanced security requirements which would involve Fortezza authorization, advanced encryption, and a multilevel secure environment.

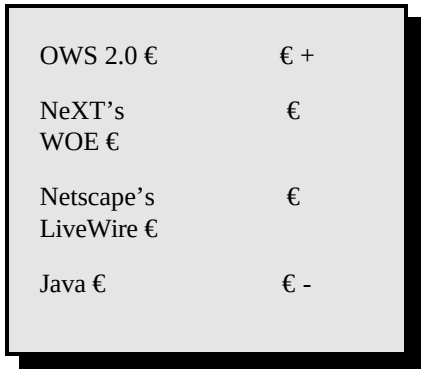
2.5.8 Fault Tolerance



Any database access solution implemented in the DII environment must have the capability to recover from catastrophic failures in any of the three logical levels discussed in Paragraph 2.2.1.2. Each of the products prototyped were capable of detecting errors when access the database and returning a status message to the user. Additionally, each product was capable of detecting an error in a transactional modification and returning the system to a stable state after the failure.

The Web itself also provides a fault tolerance environment for hosting Web based applications. Browers automatically timeout connections when responses are not received and automatically display error messages when connections cannot be established. A third fault tolerance issue is the selection of alternate database servers. Selection of an alternate database server can be presented to the user on an appropriate Web page.

2.5.9 Performance



Upon completion of the prototyping effort, SRA conducted several performance tests that measured the relative performance of the several prototype functions. The functions selected represented the functionality necessary to perform daily system operations. The performance tests were conducted for each of the previously prototyped products. SRA conducted these tests in the network environment shown earlier in Figure 2-13 which emulates the SIPRNET. SRA conducted the following specific tests:

- Loading the Java applet. An important aspect that SRA wanted to investigate was the amount of time it actually took to load the Java applet from the server to the client. Additionally, even though the same Java GUI applet was used for the OWS, WOE, and LiveWire Pro, SRA performed tests for downloading each GUI.

- Adding an air carrier to the database. SRA recorded the amount of time it actually took to update the GCCS Core Database with user supplied air carrier information. This test gives an idea of how much time it takes to make updates to multiple GCCS/JOPES Core Database tables.

- Geoloc Search queries. SRA wanted to examine the amount of time it took to perform searches for geolocation codes based on user supplied information and to return them to the user. Recall that a different method was used for each product to perform these searches. With the OWS, SRA implemented the concept of paging to return the data, but the query had to be redone each time the user requested a subsequent page. With WOE, SRA was able to use state management to implement paging that did not require the query to be redone each time the user requested the next page of data. With LiveWire, SRA did not implement paging of any kind. Instead, the data generated by a query was returned all at once to the client. As a result, in order to make comparisons among the products, SRA performed two types of queries: queries that used paging to return search results and queries that returned the entire search results at once. For queries that used paging, SRA returned 15 records at a time. SRA did not include LiveWire in these tests. For queries that did not use paging, SRA returned all the records at once. SRA tested OWS, WOE and LiveWire for these tests. For OWS and WOE, which were implemented using paging, SRA set the page size to be greater than the number of records produced by the query so that all the records were returned to the client at once. In this way, it is possible to compare solutions that use paging, i.e., OWS and WOE, with solutions that do not use paging, i.e., LiveWire. Table 2-8 summarizes the queries SRA performed.

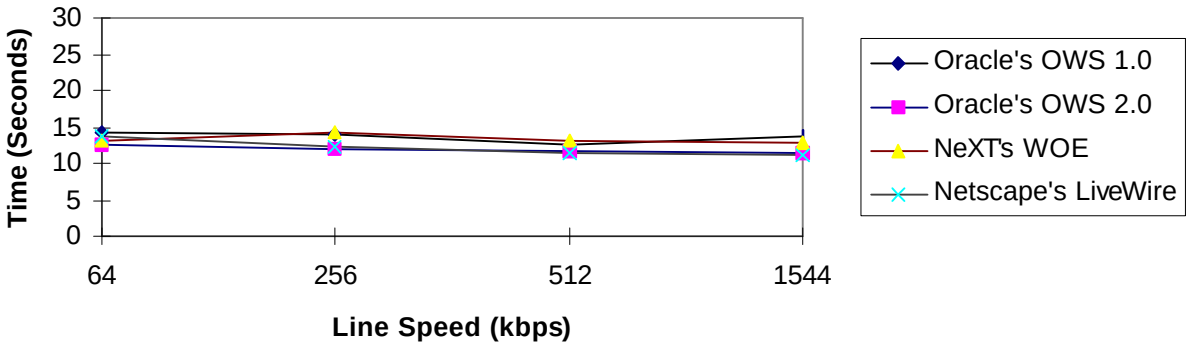


Table 2-8: Geoloc Search Queries.

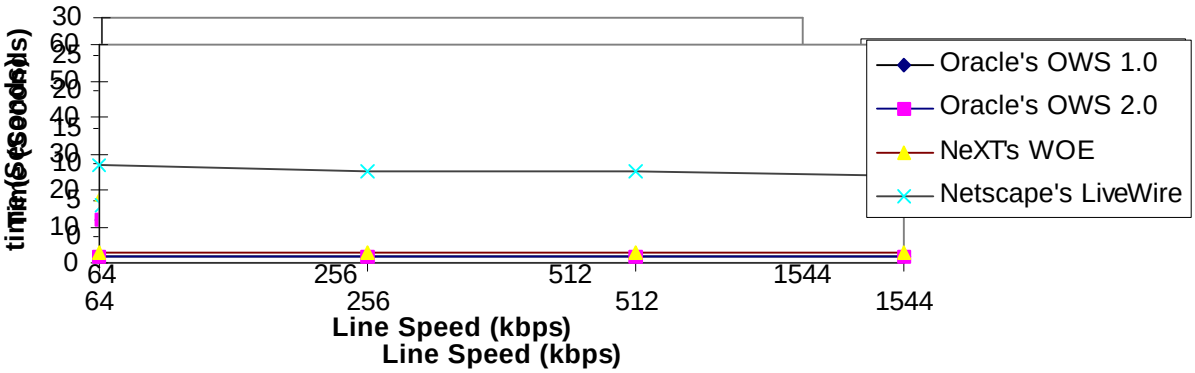
Query	Number of Records (Rows) Produced by Query	Paging	Number of Records (Rows) Returned at Once
All geolocation codes that begin with FD	181	No	181
All geolocation names that begin with FD	1	No	1
All geolocation codes that are in the country/state specified by GE	4,009	Yes	15
All geolocation codes that begin with A that are in the country/state specified by GE	283	Yes	15

An additional point about these tests is that SRA tried to eliminate external factors from influencing the tests. To this end, SRA performed the tests after hours when network traffic was minimal. Furthermore, SRA monitored the client's and the server's CPU usage to ensure that other processes were not either machine's resources.

2.5.9.1 Performance Results. SRA expected the amount of time required to download the applet would be independent of the back end database access tool being used, and this fact was demonstrated by the performance results shown in Figure 2-16.

Figure 2-16: Loading the Java Applet.

Another important point about Figure 2-16 is that the time to download the Add Air Carrier applet was not greatly affected by the available bandwidth. This observation needs to be put into context by understanding that SRA's Add Air Carrier applet is not very large in terms of byte size.



For larger applets, SRA predicts that the amount of available bandwidth will become a more important factor in achieving acceptable response times.

Figures 2-17 to 2-23 show direct comparisons for the different Add Air Carrier threads implemented using Oracle's OWS, NeXT's WOE, and Netscape's LiveWire. Figure 2-17 summarizes performance for actually adding an air carrier to the GCCS/JOPES Core Database. Figures 2-18 and 2-19 show results for tests that do not use paging so that all the products may be compared to one another. Figures 2-20 through 2-23 show results for tests that use paging in order to emphasize some important points. Note that Figures 2-20 through 2-23 do not include results for Netscape's LiveWire because paging was not implemented with this solution. Appendix C contains the actual performance number obtained for all the tests.

Figure 2-17: Adding an Air Carrier to the GCCS/JOPES Core Database.

Figure 2-18: Geoloc Search for all Geoloc Names that Begin with FD.

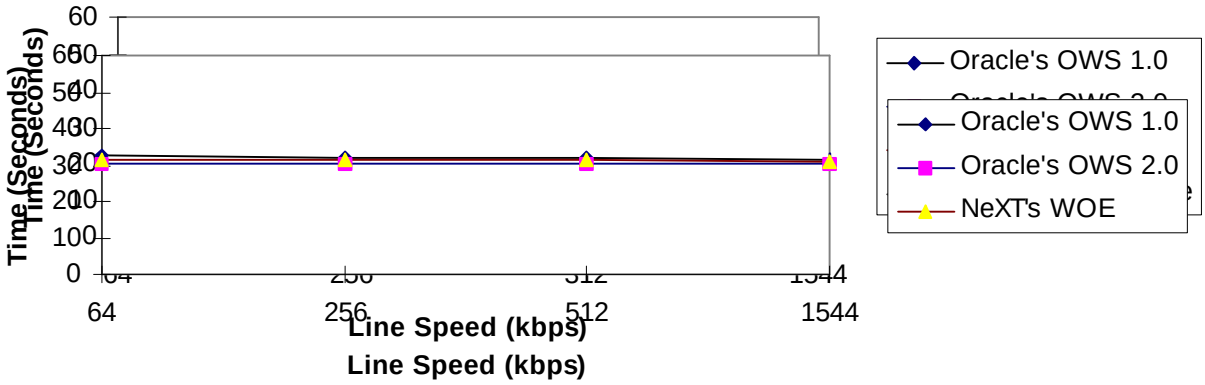


Figure 2-19: Geoloc Search for all Geoloc Codes that Begin with FD.

Figure 2-20: Geoloc Search for all Geoloc Codes in Country/State Specified by GE (First Page).

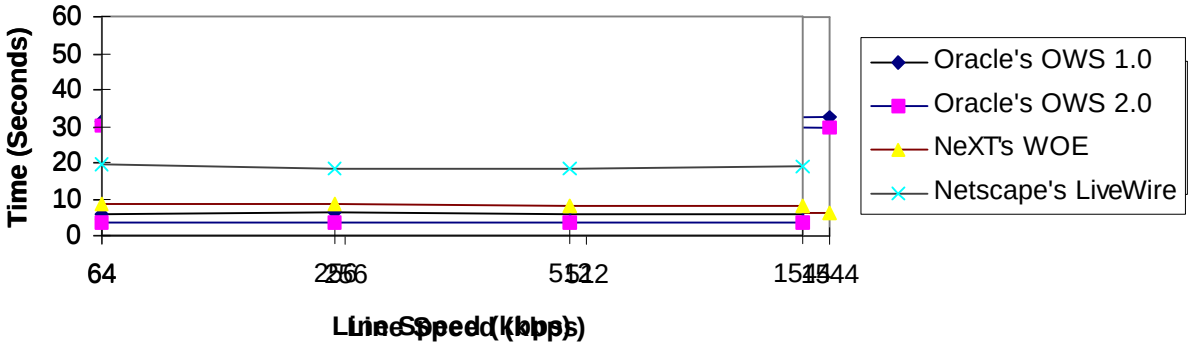


Figure 2-21: Geoloc Search for all Geoloc Codes in Country/State Specified by GE (Second Page).

Figure 2-22: Geoloc Search for all Geoloc Codes in Country/State Specified by GE that Begin with A (First Page).

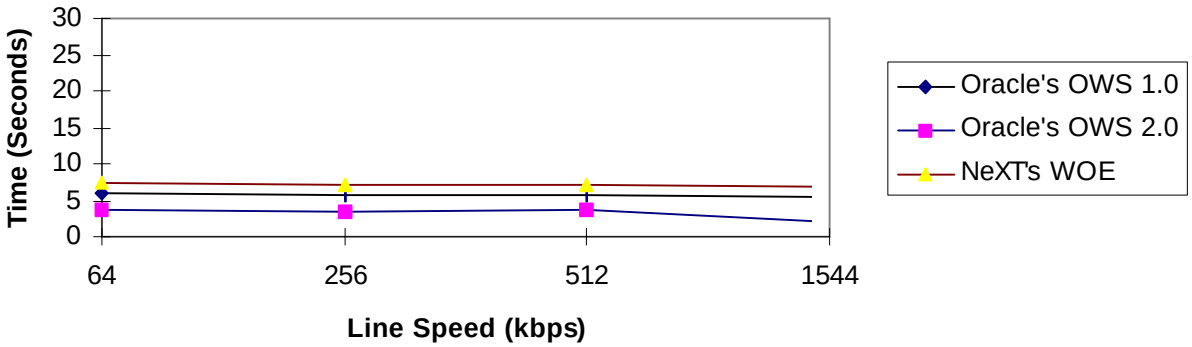


Figure 2-23: Geoloc Search for all Geoloc Codes in Country/State Specified by GE that Begin with A (Second Page).

From these performance results, SRA can make the following observations:

- Oracle's OWS is generally more performant than Netscape's LiveWire and NeXT's WOE. Additionally, OWS 2.0 improves upon the performance of OWS 1.0. SRA expected this behavior because OWS makes use of Oracle's WRB API which is more performant than the CGI used by OWS 1.0.

- A notable exception to OWS 2.0's performance dominance occurs for queries that include paging and produce a substantial number of records. Recall that although the OWS 2.0 can perform paging, it implements it by redoing the query each time a page is requested by the user. NeXT's WOE, however, supports state management for cursors, allowing paging to be implemented without redoing the query each time. Figures 2-20 and 2-21 illustrate this disparity clearly. In Figure 2-20, the query for all geolocation codes in GE is being performed for the first time, and the performance numbers for OWS 2.0 and for WOE are comparable, with OWS 2.0 being slightly faster. Now, note in Figures 2-21 that when the second page of data is returned, NeXT's WOE is considerably faster, four to five times, than OWS 2.0 because WOE supports state management for cursors.

- SRA thought that the query for all geolocation codes in GE that begin with A would demonstrate behavior similar to the query for all geolocation codes in GE, but Figures 2-22 and 2-23 show that this behavior was not the case. When the query is first performed, OWS 2.0 is slightly faster than WOE as before. When the next page of data is returned, however, OWS 2.0 is still faster than WOE. The reason for this behavior is simple. This query only produces 283 rows, while the previous query produces 4,009 rows. As a result, the amount of time it actually takes to perform this query is a smaller portion of the overall response time, and OWS is able to perform the query faster than WOE anyway. As a result, for this smaller query, OWS is able to overcome its inability to manage state for cursors, and perform faster than WOE

even though OWS redoes the query each time a user requests a page. It must be pointed out, however, that by redoing the query each time, OWS uses more server resources than WOE. SRA did not quantify this usage.

- Netscape's LiveWire is generally the least performant solution in terms of response time. For the queries without paging, LiveWire is substantially slower than OWS and WOE as shown in Figures 2-18 and 2-19. SRA did not implement paging with LiveWire because it would have required a large amount of custom code.

- In the case of adding an air carrier, the differences among the products is on the order of two seconds with NeXT's WOE being the slowest and Oracle's OWS 2.0 once again the fastest. It is SRA's opinion, however, that none of the products took an unreasonable amount of time to perform this set of business rules and data I/O.

- In general, the network bandwidth did not affect the response times. With that said, SRA would like to stress that these tests are only for the Add Air Carrier thread of S&M. Other threads, applications or even queries may be more sensitive to bandwidth changes.

- It must be remembered that these tests were run against an Oracle database. Results for accessing a Sybase or Informix database may be completely different. SRA suspects that OWS 2.0 would have a lower level of performance because it uses a gateway to access these databases.

In addition to performing the previous tests, SRA ran some tests that returned all the geolocation codes in the country/state specified by GE at one time instead of using paging. These tests were inconsistent with results ranging minutes apart for the same product at the same speed. SRA investigated this occurrence, and found that in these instances, the client's browser process was using 100 percent of the client's CPU resources for the majority of the test time. This indicates that the Java applet which is running within the confines of the browser's memory address space is using these resources. This fact points to some serious Java performance problems. Unfortunately, SRA did not have time to delve into this matter further.

2.5.10 Cost of Development and Maintenance

OWS 2.0	
NeXT's WOE	-
Netscape's LiveWire	

Development and maintenance costs are important factors in the life cycle cost of the products evaluated. We used our own experience in developing the prototypes to judge expected development costs. To estimate maintenance costs we examined the complexity of the code and the ease with which a developer with no prior knowledge of the application could modify the code.

The functionality and amount of error checking that was implemented by the SRA developers differed slightly among the prototypes. To account for this, we identified a common set of functionality in all the prototypes and determined the amount of code required to implement it. Then

using the actual amounts of code and staff-days expended, we normalized the labor for each prototype. This gave us a uniform method of measuring development cost, see Table 2.9, Comparison of Development Cost. The method yielded normalized development times for the Netscape and Oracle prototypes that were roughly equal at under 20 staff-days. The NeXT prototype took 50 staff-days for comparable functionality.

Table 2.9 Comparison of Development Cost.

Source Lines Of Code (SLOC)	Oracle WebServer	NeXT WebObjects	Netscape LiveWire
SLOC for common functionality	965	1873	718
Actual SLOC	1250	1873	436
Actual Staff-days	20	50	10
Normalization Factor	0.77	1.0	1.65
Normalized Staff-days	15.4	50.0	16.5

A comparison of the development costs would not be complete without the developers observations . The OWS development environment provided a simple to use tool kit that permitted the development of the Oracle prototype to proceed with a minimal learning curve. Excellent run-time error messages from the Oracle Web Agent greatly aided debugging. The JavaScript language used by Netscape's LiveWire also proved to be a simple environment to work with. It is a relatively small language with a simple object model. However, because the language is loosely type, permits the compiler to allow simple coding errors that required additional iterations of the code-test-debug cycle.

Of the three development environments, NeXT's WOE was the most difficult. A steeper learning curve, greater complexity, lack of informative run-time error messages and the documentation quality were factors that made WOE difficult to use. WOE required the developers to work using an object-oriented approach. This offers the promise of creating more reusable software. It can lower development costs in the long run by making it easier to spread the cost of developing common code across several applications. It is expected that as the pool of developers with object-oriented experience increases the learning curve will become less of a factor.

The evaluation of the maintenance costs is more subjective because all three products are new and no large scale applications, developed with the products have been fielded. To evaluate the products scenarios are considered, a change in database structure, and a modification to a business rule.

WOE uses an abstract, relatively data-independent database interface. This will make implementing database structure changes easier than with OWS or LiveWire. PL/SQL and JavaScript programmers often directly refer to database tables and data items. Users of NeXT's EOF are less likely to do so.

Changes to a business rule would likely be implemented in WOE as a change to a business object possibly shared by multiple applications. A change to a business rule in JavaScript or PL/SQL might cause several modules to have to be modified, if the rule is used by more than one application or user function.

Overall, for sufficiently large distributed systems with considerable up-front investment and complex business rules, WOE may hold an advantage in maintainability over OWS and LiveWire. For smaller applications, the simplicity of OWS and LiveWire should result in lower maintenance costs.

2.6 CONCLUSIONS AND RECOMMENDATIONS

Through the course of performing this evaluation SRA has formulated several conclusions. First, this section presents conclusions concerning the specific technologies discussed in this paper and then conclusions concerning web technology in general.

2.6.1 OWS

The OWS 2.0 demonstrated several strengths during the prototype effort. These strengths include:

- Excellent database access capabilities for Oracle databases. Oracle's use of PL/SQL provides programmers with an extensive ability to access data residing in Oracle databases.

- Completeness of product. The OWS is in its second release, meaning that Oracle has had the opportunity to make improvements to the first release. Furthermore, SRA did not encounter errors, or bugs, in the OWS prototype effort. Also, Oracle made extensive documentation available to SRA that greatly assisted SRA's effort. Additionally, Oracle is working on its OWS 3.0 which will add additional important functionality such as intercommunication between WRB cartridges and state management for cursors.

- Performance. The OWS 1.0 was more performant than the Netscape and NeXT products, and the OWS 2.0 improves upon this performance with the use of its WRB API. With the addition of more state management capabilities, the OWS 3.0 will be able to address the concept of paging for queries that return substantial amounts of records.

- € Product support. It was obvious to SRA that Oracle is committed to supporting the OWS and offered SRA excellent support during the prototype effort

Weaknesses of OWS 2.0 observed by SRA during this evaluation include:

- € Process distribution. Both OWS 1.0 and 2.0 are dependent upon PL/SQL to implement the data I/O layer. This dependency complicates process distribution and minimizes it efficacy by necessitating the use of an Oracle database to act as a storage and execution space for the PL/SQL stored procedures. Oracle recognizes its dependency on its own database as a potential limitation and is planning to deviate from this paradigm in its OWS 3.0 Version.

- € Configuration management. Although PL/SQL stored procedures can execute on several different OS and hardware platforms without having to be recompiled, as already stated, they require the presence of an Oracle database. This database increases the burden of configuration management.

- € Interoperability. While OWS 2.0 satisfies the minimal requirement of interoperating with the Netscape Web browser, SRA still considers it a weakness that OWS 2.0 requires the Oracle Web Listener.

2.6.2 NeXT's WOE

NeXT's WOE offers several advantages for developing Web-based database application tools. These advantages include:

- € Object-oriented model. WOE allows for the map of a database schema to an object model, enabling developers to fuse data residing in a DBMS with object methods, or business rules. This ability in turn facilitates code changes and reuse.

- € State management of cursors. WOE is the only solution evaluated by SRA that provides state management for cursors. This ability is important because it allows for efficient, i.e., network and server performant, implementations of queries that return large amounts of records.

- € Product support. Although SRA did encounter some problems with WOE, NeXT was committed to solving these problems and worked closely with SRA's developers to overcome them.

- € Interoperability. NeXT Web-based application environment offered the most interoperability due to its support of any CGI capable HTTP server and its ability to interoperate with OLE/COM.

SRA made the following observations concerning WOE's weaknesses:

- Development time and cost. The WOE environment is difficult to master, requiring by far the most amount of time to develop an application. Additionally, although Objective C is based on the C programming language, Objective C is not a widely used and known language at this time, meaning there is not a large base of already existing Objective C developers.

- Product cost. Based on the information SRA was able to gather on pricing, the WOE is by far the most expensive product evaluated in this study.

- Documentation. The maturity of WOE's documentation has not caught up to the maturity of the product itself.

- Process Distribution and configuration management. WOE is only weak in this area in the sense that it relies on Objective C which must be recompiled for different platforms, complicating process distribution and configuration management.

2.6.3 Netscape's LiveWire Pro

SRA noted the following advantages of Netscape's LiveWire Pro product.

- Prototype ease of development. LiveWire Pro's development environment greatly lent itself to the development of the Add Air Carrier thread of Scheduling and Movement. In particular, JavaScript was easy to learn and allowed for rapid application development.

- Development time and cost. Due to the ease of development, LiveWire Pro leads to lower development time and cost than OWS and WOE.

- Product cost. The Netscape product suite, the HTTP server and LiveWire Pro, is substantially less expensive than WOE and notably less expensive than OWS 2.0.

SRA noted the following weaknesses associated with LiveWire Pro:

- Depth of functionality. For the same reasons that LiveWire Pro greatly facilitated the development of the prototype, SRA has concerns about LiveWire's ability to support the development of an application on a larger scale. In particular, LiveWire's use of JavaScript is a concern. JavaScript's main design goal is to allow HTML programmers to increase the functionality of their HTML pages without having to master a complex, full featured programming language. It is JavaScript's lack of support for more complex functionality that concerns SRA. Netscape's recognizes this area as a weakness and is planning to incorporate support for Java when it releases the Enterprise Server.

- Product direction and support. During the evaluation, Netscape informed SRA that LiveWire would not interoperate with the Commerce Server when it was released as a production version. This forces SRA to question how well thought out Netscape's product direction is. Additionally, Netscape's documentation is by far the most lacking of the products in this evaluation. Another cause of concern was Netscape's lack of support for SRA during the evaluation. When SRA encountered problems, SRA had to find workarounds for those problems without help from Netscape.

- State management for cursors. LiveWire Pro offers no out of the box functionality for state management of cursors. This fact hinders LiveWire's ability to implement efficiently queries that return large amounts of records.

- Performance. LiveWire was the least performant solution in terms of response times. One reason is the already stated lack of state management for cursors, but even in tests where this deficiency was not a factor LiveWire was notably less performant than OWS 2.0 and WOE.

- Interoperability. While LiveWire Pro satisfies the minimal requirement of interoperating with Netscape's Web browser, it also requires Netscape's HTTP server, i.e., Enterprise Server, because of its reliance on the proprietary NSAPI.

2.6.4 Java

Even though this evaluation's focus is on database access, SRA has developed the following conclusions concerning Java:

- Increased functionality over HTML. A major advantage of using Java instead of HTML to provide user interfaces is Java provides more GUI functionality than HTML. Java includes classes for developing GUI widgets that are not possible using HTML. Additionally, Java allows the developer to create dynamic web pages without having to serve an entirely new page from the HTTP server as is the case with HTML. Another important trait of Java is it is multi-threaded.

- Platform independence. Java uses machine independent bytecode which enables Java programs to run on any client or server with a Java interpreter, regardless of the OS and hardware platform, without being recompiled.

- Configuration management. Java applets are downloaded from the server to the client. This characteristic greatly reduces the configuration management burden because in most environments the number of clients is far greater than the number of servers. Applets can be staged on the smaller number of servers, and when changes to the applets are necessary, it is only necessary to make the changes at the server.

Weaknesses with Java include:

- Language immaturity. Java has only existed for about eight months, and as a result, it has not had the time to go through revisions to hammer out any problems with it and to add needed functionality. As an example, Gartner Group reports that Java needs to incorporate operator overloading to support new numerical data types. Additionally, because Java is a nascent language, there is not a large group of skilled Java developers. Java's designers attempted to mitigate this problem by basing it on C and C++ which does have a large population of experienced developers.

- Lack of third party Java tools. Due to Java's recent creation, there are not a lot of third party tools that facilitate Java development. Additionally, many of the tools that are available are first releases or beta products, meaning they have problems that must be solved. SRA did not extensively study this area, but this fact is readily apparent.

- Performance. As explained in the performance section, SRA has concerns about Java's ability to support mission critical applications. SRA's performance testing is not sufficient to make this claim outright, but Gartner group supports this hypothesis, saying that Java's ". . . performance may not be acceptable for mission critical applications."

- Security. Java includes many internal security mechanisms, but this area has to be considered a risk until the language has been used more and subjected to more scrutiny.

- Inconsistent look for different browsers. This is another problem associated with the newness of Java. Some browsers, even from the same company, paint Java widgets differently due to the fact that their Java interpreters are not identical and some result in errors.

2.6.5 General WWW Conclusions

The WWW technology landscape is rapidly changing, and new products are continually emerging. For example, during SRA's study, Oracle released a second version of its OWS, and Netscape moved from a Beta 2 product to a Beta 4 product. These changes are just two of many that have occurred during SRA's evaluation. Additionally, Web technology as a whole is immature. Java is a new programming language and the concept of accessing DBMSs via the Web is fairly new. SRA feels that the rapidly changing Web landscape and the immaturity of many Web products represent risk. It is important to be careful when investing in a particular technology. This investment may inextricably link the investor to a particular company's that may not even be around in a couple of years or to a technology that may be obsolete in a few years.

Deploying applications on the WWW is very much a network-centric solution, and as such, network response time becomes critical. In SRA's prototype, the amount of available bandwidth did not greatly affect performance. In particular, the amount of time it took to download the Add Air Carrier thread did not increase significantly at lower bandwidths. This behavior has to be

tempered with the knowledge that the Add Air Carrier GUI applet is only approximately 64.5 kilobytes in size. The point is that as Java applets become more complex and encompass more functionality their size will also increase, making available bandwidth a more crucial parameter.

Another potential problem area when deploying applications on the WWW is security even if the network on which the application is deployed is an Intranet. There is a long list of security mechanisms available to an administrator such as encryption at the hardware and software levels and client authentication. In order for these mechanisms to be effective, however they have to be properly installed and administered which is not a trivial task.

To this point, SRA has addressed concerns pertaining to deploying an application on the WWW. There are of course many benefits with doing this as well. The Web has the potential to bring completely platform independent applications to the desktop. This potential is already largely filled in terms of clients. As platform independent languages like Java or JavaScript become more widely used and supported by HTTP servers' APIs, this potential will also be fulfilled at the server. Not only does this platform independence facilitate the idea of distributing processing to where it makes the most sense, it also tremendously eases the configuration management workload.

2.6.6 Recommendations

As a result of performing this evaluation, SRA has formulated several recommendations. First, SRA does not believe that the state of Web technology is ready to field mission critical applications for use by power users. By power users, SRA means users who use an application heavily as the primary aspect of their job. Instead, SRA feels that the current state of Web Technology lends itself to providing application functionality, especially database browsing, for the casual user. For heavy users of S&M, SRA feels the performance that can be provided by current Web backend database solutions is not at the level it needs to be and the level of functionality that can be provided by a GUI developed in Java and HTML is not sufficient for people who use an application regularly. This does not mean that it is not a good idea to begin the migration of GCCS Database application threads. On the contrary, now is a perfect time to become more knowledgeable about the details of deploying applications on the WWW. For example in the case of Java, Gartner Group predicts that for at least the next 18 months, companies will be better served to concentrate on cultivating Java programming expertise rather than trying to implement mission critical business applications in Java.

Programming expertise is not SRA's primary concern with Java. Instead, SRA is most concerned about Java's performance and security. Sun claims that for most applications, Java provides more than adequate processing and performs only slightly worse than machine specific code. SRA feels that these two areas must be investigated more thoroughly in order to substantiate this claim. SRA's performance testing, although by no means a complete analysis of Java's performance, indicate that Java may have some serious performance problems when processing large amounts of data. Java's security needs to be investigated further as well. Already, in the short time it has been out, security holes have been discovered. Although most of these resulted from improper enforcement by the browser's Java interpreter, the fact remains that Java is susceptible to security breaks.

Other areas that SRA feels need to be evaluated more in depth include HTTP servers and Web browsers. HTTP servers play a significant role in terms of performance and security for the overall system. The current widely used API is the CGI, but this interface lacks the performance to support complex applications and high transaction rates. HTTP vendors are developing their own proprietary more performant APIs such as Netscape's NSAPI and Oracle's WRB API. It is important to study this emerging trend from a technical and business standpoint in order to choose the most appropriate API in which to invest. HTTP security mechanisms must be studied and tested to determine which ones will provide the best level of security. In the case of Web browsers, SRA recommends that the Government study the developing market of plug-ins and helper applications that third party vendors are developing for Web browsers. In this way, the idea of using COTS products may be advanced. Additionally, if more advanced security features are desired, it must be ensured that the Web browser can support them. Another point that relates to Java but pertains to both HTTP servers and Web browsers as well is the ability to cache Java applets to improve performance. This concept should be investigated and tested.

SRA wants to stress that new web application development tools have been developed during SRA's study and more will be developed in the future. With this said, however, SRA feels that the direction the government should take is to monitor the progress of and eventually evaluate 4GLs tools that provide the ability to develop code using C, C++ and other hardware and OS specific programming languages for power users who need better performance and interface quality and the ability to develop code using platform independent languages such as Java for the larger number of casual users. The latter applications may be less performant than the former, but they will facilitate process distribution and especially configuration management. At a minimum, if the government is to proceed with Java development, it must study Java development tools to facilitate the process. As an accompanying thought, SRA would also like to point out the emergence of middleware products that can convert hardware and OS specific code such as Ada into platform independent bytecode. This area should be studied as well.

APPENDIX A

ACRONYMS

APPENDIX A - ACRONYMS

4GL . . . . . Fourth Generation Language

ANSI . . . . . American National Standards Institute
API . . . . . Application Programming Interface
ASCII . . . . . American Standard Code for Information Interchange
AWT . . . . . Abstract Windows Toolkit

CGI . . . . . Common Gateway Interface
CLI . . . . . Call Level Interfaces
COE . . . . . Common Operating Environment
COM . . . . . Component Object Model
CORBA . . . . . Common Object Request Broker Architecture
COTS . . . . . Commercial-Off-the-Shelf
CPU . . . . . Central Processing Unit

DBD . . . . . Database Distribution
DBMS . . . . . Database Management System
DCE . . . . . Distributed Computing Environment
DII . . . . . Defense Initiative Infrastructure
DISA . . . . . Defense Information System Agency

E/R . . . . . Entity/Relationship
EOF . . . . . Enterprise Objects Framework
EO Modeler . . . . . Enterprise Objects Modeler
ESQL . . . . . Embedded Structured Query Language

FTP . . . . . File Transfer Protocol

GCCS . . . . . Global Command and Control System
GIF . . . . . Graphical Interchange Format
GUI . . . . . Graphical User Interface

HTF . . . . . HyperText Function
HTML . . . . . HyperText Markup Language
HTP . . . . . HyperText Procedure
HTTP . . . . . HyperText Transport Protocol

IDE . . . . . Integrated Development Environment
I/O . . . . . Input/Output
IP . . . . . Internet Protocol

JOPES . . . . . Joint Operations Planning and Execution System
JPEG . . . . . Joint Photographic Experts Group

kbps . . . . . kilobits per second

LAN . . . . . Local Area Network

MAC . . . . . Message Authentication Code
Mbps . . . . . Megabits per second

NSAPI . . . . . Netscape Application Programming Interface

OCI . . . . . Oracle Call Interface
ODBC . . . . . Open Database Connection
OLE . . . . . Object Linking and Embedding
OPLAN . . . . . Operational Plan
OS . . . . . Operating System
OWA . . . . . Oracle Web Agent
OWS . . . . . Oracle WebServer

PDO . . . . . Portable Distributed Objects
PL . . . . . Procedural Language

S&M . . . . . Scheduling and Movement
SDK . . . . . System Development Kit
SIPRNET . . . . . Secret Internet Protocol Router Network
SQL . . . . . Structured Query Language
SRA . . . . . Systems Research and Applications
SRS . . . . . System Requirements Specification
SSL . . . . . Secure Sockets Layer

TCP . . . . . Transport Control Protocol

URL . . . . . Uniform Resource Locator

WAIS . . . . . Wide Area Information Search
WAN . . . . . Wide Area Network
WOE . . . . . WebObjects Enterprise
WOF . . . . . WebObjects Framework
WRB . . . . . Web Request Broker
WRBX . . . . . Web Request Broker Executable
WWMCCS . . . . . World Wide Military Command and Control System
WWW . . . . . World Wide Web
WYSIWYG . . . . . What-You-See-Is-What-You-Get

APPENDIX B

BIBLIOGRAPHY

APPENDIX B - BIBLIOGRAPHY

A Guide To URLs at http://www.netspace.org/users/dwb/url-guide.html. David W. Baker.

HTML Publishing on the Internet. Brent Heslop and Larry Budnick. 1995.

Object-Oriented Programming and the Objective C Language. NeXT Computer Inc. 1990-1993.

"Teach Yourself Java in 21 Days", Laura Lemay and Charles L.Perkins, Sams Net 1996.

Reference pages from Sun, URL -> java.sun.com/java.sun.com/JDK-1.0/api/packages.html.

Java and the New Wave of Internet Programming Languages. Ed Yourdon, Corporate Internet Strategies January 1996, vol II, no 1, p1-16.

Java & HotJava: Waking up the Web. Sean Gonzalez PC Magazine October 1995 v14 n18 p265-268.

The Java Language: A White Paper. Sun Microsystems.

Java Tools IDE's at a glance -> www.rsi.com/info/java-info.html.

Enterprise Objects Framework Developer's Guide, NextStep Developer's Library Release 3. NeXT Computer, Inc. 900 Chesapeake Drive, Redwood City, CA 94063.

Object-Oriented Programming and the Objective C Language, NextStep Developer's Library Release 3. NeXT Computer, Inc. 900 Chesapeake Drive, Redwood City, CA 94063.

Oracle WebServer User's Manual, Release 1.0.1, Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle WebServer 2.0 Technical Notes. Oracle Corporation. March 1996.

"Netscape Commerce Server Data Sheet" at http://home.netscape.com. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043.

"The Netscape Server API" at hhttp://home.netscape.com. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043.

"Netscape Navigator LiveWire and LiveWire Pro Data Sheet" at http://home.netscape.com. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043.

"Netscape Enterprise Server Data Sheet" at http://home.netscape.com. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043.

"Netscape Enterprise Server FAQ" at http://home.netscape.com. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043.

"LiveWire Developer's Guide" at http://home.netscape.com. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043.

"Getting Started". NeXT Computer, Inc. 900 Chesapeake Drive, Redwood City, CA 94063.

"Using WebScript". NeXT Computer, Inc. 900 Chesapeake Drive, Redwood City, CA 94063.

Netscape Communications Server Programmer's Guide. Netscape Communications Corporation, 501 E. Middlefield Rd., Mountain View, CA 94043. 1995.

"Architecting for Change with the Enterprise Objects Framework and Portable Distributed Objects." May 1995. NeXT Computer, Inc. 900 Chesapeake Drive, Redwood City, CA 94063.

"Is Java Ready for In-House Brewing?" N. Jones and M. West. 10 April 1996. GartnerGroup.

"Web Server APIs: Challenging the Ubiquitous CGI, Part 1" D. Plummer. 29 April 1996. GartnerGroup.

SRA Corporation. JOPES Database Distribution (DBD) Alternatives Study. 15 May 1995.

SRA Corporation. Distributed Computing Analysis Report. 6 February 1996.

Oracle WebServer Installation Guide for Sun SPARC Solaris 2.x, Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle WebServer User's Guide, Release 2.0, Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

"Three-Tier Computing Architectures and Beyond". R. Schulte. GartnerGroup. 25 August 1995.

APPENDIX C

PERFORMANCE TESTING DATA

APPENDIX C - PERFORMANCE TESTING DATA

	Line Speed			
	64kbps	**256kbps**	**512kbps**	**1544kbps**
Loading the Java Applet Test 1				
Oracle's OWS 1.0	13.94	14.07	12.5	13.9
Oracle's OWS 2.0	12.56	12.03	12.35	11.56
NeXT's WOE	13.25	13.78	12.93	13.44
Netscape's LiveWire	15.94	11.88	11.15	11.12
Loading the Java Applet Test 2				
Oracle's OWS 1.0	14.5	13.78	12.38	13.53
Oracle's OWS 2.0	12.47	12.11	11.19	11.56
NeXT's WOE	13.28	14.81	13.19	12.38
Netscape's LiveWire	15.31	12.44	11.53	10.94
Loading the Java Applet Average				
Oracle's OWS 1.0	14.22	13.93	12.44	13.72
Oracle's OWS 2.0	12.51	12.07	11.77	11.56
NeXT's WOE	13.27	14.3	13.06	12.91
Netscape's LiveWire	13.62	12.16	11.34	11.03
Adding an Air Carrier to the GCCS/JOPES Core Database Test 1				
Oracle's OWS 1.0	2.53	2.81	2.47	2.31
Oracle's OWS 2.0	2.44	2.59	2.53	2.1
NeXT's WOE	5.56	5.5	5.56	5.96
Netscape's LiveWire	4.41	3.13	3.25	2.82
Adding an Air Carrier to the GCCS/JOPES Core Database Test 2				
Oracle's OWS 1.0	2.31	2.34	2.4	2.35
Oracle's OWS 2.0	2.34	2.06	2.31	2.11
NeXT's WOE	5.82	4.59	5.22	5.54

	Line Speed			
	64kbps	**256kbps**	**512kbps**	**1544kbps**
Netscape's LiveWire	4.93	4	3.09	2.97
Adding an Air Carrier to the GCCS/JOPES Core Database Average				
Oracle's OWS 1.0	2.42	2.58	2.44	2.33
Oracle's OWS 2.0	2.39	2.32	2.42	2.1
NeXT's WOE	5.69	5.04	5.39	5.75
Netscape's LiveWire	4.17	3.57	3.17	2.9
Geoloc Search for all GEOLOC Names that Begin with FD Test 1				
Oracle's OWS 1.0	1.65	1.56	1.96	1.47
Oracle's OWS 2.0	1.6	1.82	1.6	1.91
NeXT's WOE	3.09	3.03	2.81	2.82
Netscape's LiveWire	26.12	24.32	25.25	23.47
Geoloc Search for all GEOLOC Names that Begin with FD Test 2				
Oracle's OWS 1.0	2	2.28	1.78	1.69
Oracle's OWS 2.0	1.6	1.44	1.41	1.9
NeXT's WOE	3.06	3.03	2.81	2.94
Netscape's LiveWire	27.25	25.97	25.25	24.12
Geoloc Search for all GEOLOC Names that Begin with FD Test Average				
Oracle's OWS 1.0	1.83	1.92	1.87	1.58
Oracle's OWS 2.0	1.6	1.63	1.5	1.9
NeXT's WOE	3.08	3.03	2.81	2.88
Netscape's LiveWire	26.69	25.15	25.25	23.8
GEOLOC Search for all GEOLOC Codes that Begin with FD Test 1				
Oracle's OWS 1.0	10.87	10.34	10.5	10.25
Oracle's OWS 2.0	10.34	9.81	10.37	10.44
NeXT's WOE	19.44	18.94	18.92	18.31
Netscape's LiveWire	33.78	32.06	34.69	31.44

	Line Speed			
	64kbps	**256kbps**	**512kbps**	**1544kbps**
GEOLOC Search for all GEOLOC Codes that Begin with FD Test 2				
Oracle's OWS 1.0	12.18	10.53	11.38	10.38
Oracle's OWS 2.0	9.81	10.13	10.59	10.16
NeXT's WOE	21.78	19	18.79	18.25
Netscape's LiveWire	32.59	31.19	31.31	29.69
GEOLOC Search for all GEOLOC Codes that Begin with FD Test Average				
Oracle's OWS 1.0	11.53	10.44	19.94	10.32
Oracle's OWS 2.0	10.07	9.97	10.48	10.3
NeXT's WOE	20.61	18.97	18.86	18.28
Netscape's LiveWire	33.19	31.63	33	30.57
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE (1st Page) Test 1				
Oracle's OWS 1.0	32.03	31.75	31.85	31.19
Oracle's OWS 2.0	30.68	30.04	30.31	30
NeXT's WOE	32.35	31.47	30.88	31.5
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE (1st Page) Test 2				
Oracle's OWS 1.0	33.31	31.91	32.69	31.75
Oracle's OWS 2.0	30.12	30.08	30.55	30.72
NeXT's WOE	30.72	30.88	31.69	29.79
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE (1st Page) Test Average				
Oracle's OWS 1.0	32.67	31.83	32.27	31.47
Oracle's OWS 2.0	30.4	30.06	30.43	30.36
NeXT's WOE	31.56	31.18	31.29	30.65
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE (2nd Page) Test 1				

	Line Speed			
	64kbps	**256kbps**	**512kbps**	**1544kbps**
Oracle's OWS 1.0	31.85	33.9	31.93	32.35
Oracle's OWS 2.0	30.36	30.34	29.78	29.86
NeXT's WOE	6.62	6.91	6.19	5.94
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE (2nd Page) Test 2				
Oracle's OWS 1.0	31.03	31.88	31.57	32.5
Oracle's OWS 2.0	30.14	29.91	30.67	29.95
NeXT's WOE	6.5	6.22	5.97	6.13
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE (2nd Page) Average				
Oracle's OWS 1.0	31.44	32.89	31.75	32.43
Oracle's OWS 2.0	30.25	30.13	30.23	29.91
NeXT's WOE	6.56	6.57	6.08	6.04
GEOLOC Search for all GEOLOC Codes in Country/State Specified by GE that begin with A (1st Page) Test 1				
Oracle's OWS 1.0	5.85	6.63	5.72	5.47
Oracle's OWS 2.0	3.62	3.44	3.28	3.34
NeXT's WOE	9.15	8.25	7.93	7.81
Netscape's LiveWire	18.9	18.46	18.97	18.5
GEOLOC Search for all GEOLOC Codes in Country/State Specified by GE that begin with A (1st Page) Test 2				
Oracle's OWS 1.0	5.66	6	5.97	5.54
Oracle's OWS 2.0	3.47	3.21	3.05	3.56
NeXT's WOE	8.5	8.4	8.03	7.88
Netscape's LiveWire	19.72	18.22	18.12	19.03
GEOLOC Search for all GEOLOC Codes in Country/State Specified by GE that begin with A (1st Page) Average				
Oracle's OWS 1.0	5.76	6.3	5.85	5.51

	Line Speed			
	64kbps	**256kbps**	**512kbps**	**1544kbps**
Oracle's OWS 2.0	3.6	3.33	3.17	3.45
NeXT's WOE	8.83	8.33	7.98	7.85
Netscape's LiveWire	19.31	18.34	18.55	18.76
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE That Begin With A (2nd Page) Test 1				
Oracle's OWS 1.0	5.66	6	5.97	5.54
Oracle's OWS 2.0	3.78	3.48	3.67	3.72
NeXT's WOE	8.5	8.4	8.03	7.88
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE That Begin With A (2nd Page) Test 2				
Oracle's OWS 1.0	6.1	5.69	5.57	5.59
Oracle's OWS 2.0	3.45	3.33	3.78	3.18
NeXT's WOE	6.53	6.03	6.13	6
GEOLOC Search for All GEOLOC Codes in Country/State Specified by GE That Begin With A (2nd Page) Average				
Oracle's OWS 1.0	5.88	5.85	5.77	5.57
Oracle's OWS 2.0	3.62	3.46	3.73	1.95
NeXT's WOE	7.52	7.22	7.08	6.94

APPENDIX D

KNOWN JAVA ERRORS

APPENDIX D - KNOWN JAVA ERRORS

Java Developers Kit 1.0.2, Known Bugs

The known bugs are divided into the following categories:

* Runtime
* Applet Viewer
* Compiler
* Class Libraries
    - o AWT (java.awt.*)
    - o IO (java.io.*)
    - o Net (java.net.*)
    - o Util (java.util.*)

Runtime

Programs that start multiple threads will not exit under Windows 95 or Windows NT.
Workaround: You can workaround this problem by either calling System.exit(int exitCode) in the last thread running or by monitoring the threads by calling Thread.join(). See java.lang.System or java.lang.Thread for more information. (1234318)

Ordered comparisons with NaN do not return false (Microsoft Windows 95 and Windows NT) Greater than, less than, greater than or equal, or less than or equal comparisons with Not A Number (Float.NaN or Double.NaN) should always return false. Instead Not A Number evaluates as less than NEGATIVE_INFINITY. (1240029)

Conversions from double to integer for very large double values, such as Double.MAX_VALUE, should produce Integer.MAX_VALUE. (1240807)

Conversions from double to long for very large double values, such as Double.MAX_VALUE, should produce Long.MAX_VALUE. (1240810)

The AWT.DLL expects at least 256 colors available for display. If you attempt to start a Java windowing program, such as Appletviewer, on a computer that cannot display 256 colors, the program will crash in the AWT.DLL module.
Workaround: Set your display to at least 256 colors. (1240271)

Applet Viewer

The Applet Viewer tag display dialog box does not display the HTML Applet tag width and height values, instead it displays the applet's actual width and height. (1231235)

Running the Applet Viewer in debug mode, file URLs currently throw access violation exceptions or NumberFormatExceptions when referenced outside the current working directory by the debugger.

To work around this problem, either set your working directory to the directory which has the class to be debugged (the classes should be in the current directory), or reference the applet from an http URL. (1231289)

Compiler

The javac compiler cannot distinguish between the same class name in different packages. So the following code example will return a compiler error:

```
package pkg;
    import java.util.*;

    public class Vector
    {
        private int i;
        public Vector()
        {
           i = 1;
        }
    }
```

(1231283)

The javac compiler does not distinguish between class names that are the same except for the case of the name. So compiling the following code example will return a compiler error:

```
    /* file CompilerTest.java */
    class CompilerTest
    {
      // class definition here
    }
   /* file compilertest.java */
    class compilertest
    {
      // class definition here
      void function(CompilerTest ct)
      {
        // do some work here
      }
    }
```

(1230037)

The javac compiler will not compile any method with more than 63 local variables. The minimum acceptable upper bound on local variables has yet to be defined. (1240530)

Class Libraries

* Abstract Window Toolkit (java.awt)

(None)

* Input/Output Package (java.io)

Asynchronous IO fails on the original shipping version of Sun Solaris 2.4. This is due to a known problem with Sun Solaris 2.4. A patch is available from Sun Microsystems Inc., patch number: OS patch 101945-34. (1199579)

* Net Package (java.net)

Socket.close() does not close the TCP connection. The TCP connection will close when you exit the program.(1234731)

* Util Package (java.util)

java.util.Date.setDate()

The method java.util.Date.setDate(int date) should recalculate the month, and year when month is 11, when the parameter date is greater than the number of days in the current month. java.util.Date.setDate(int date) should recalculate the day of week. (1226859, 1226864)

The method java.util.Date.setMonth(int month) should recalculate the year when the parameter month is greater than 11. java.util.Date.setMonth(int month) should also recalculate the day of week when the parameter month is greater than 11. (1229001, 1229000)

The method java.util.Date.setYear(int year) should recalculate the day of week. java.util.Date.setYear(int year) does not recalculate the month or date when going from a leap day, such as 29-Feb-1992, to a non-leap year. (1229004, 1231114)

Workarounds for these java.util.Date bugs

There is a very simple workaround for these java.util.Date bugs. Simply create a new Date object with the correct information in it.

```
/* Create new date object for 28-Feb-1996 */
Date date1 = new Date(96, 01, 28);
/* Do some work here */
/* Need to set the date to one week after this date
 * This will correctly create a new date of 06-Mar-96
 */
date1 = new Date( date1.getYear(),
                  date1.getMonth(),
                  date1.getDate() + 7,
                  date1.getHours(),
```

```
                date1.getMinutes(),
                date1.getSeconds() );
```

Date.toString() returns different results on different platforms. On Sun Solaris the result is:
Tue Jun 27 13:34:37 PDT 1995
On Microsoft Windows NT and Windows 95 the same date is converted to the following string:
Tue Jun 27 13:34:37 1995
Note that Java does not print the time zone on Windows. (1212188)